Proyecto Fin de Grado GITT

Aplicación de Path Planning de un Robot Móvil basado en Raspberry Pi

Autor: Iván Cabezas Moraga
Tutor: Daniel Gutiérrez Reina

Dept. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Grado
GITT

Aplicación de Path Planning de un Robot Móvil basado en Raspberry Pi

Autor:
Iván Cabezas Moraga

Tutor:
Daniel Gutiérrez Reina
Doctor Ingeniero en Electrónica

Dept. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Carrera: Aplicación de Path Planning de un Robot Móvil basado en Raspberry Pi

Autor: Iván Cabezas Moraga

Tutor: Daniel Gutiérrez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mis padres

Agradecimientos

En estas líneas quiero expresar mi máximo agradecimiento a mis padres cuya aportación ha sido fundamental a lo largo de mi vida para conseguir mis objetivos. También a todos los profesores (especialmente a mi tutor) que he tenido a lo largo de mi formación académica, reciban desde este documento mi agradecimiento por su esfuerzo y apoyo.

Iván Cabezas Moraga

Sevilla, 2017

Resumen

Este documento está compuesto de dos partes: una teórica y otra práctica. La primera tiene como objeto proporcionar una introducción a la robótica abordando los tipos de robots y sus aplicaciones. Asímismo, se trata un enfoque al problema de optimización y una introducción al problema de transporte, explicando los tipos de grafos más comunes y su tratamiento mediante los distintos tipos de algoritmos.

La segunda parte aborda el montaje de un vehículo robot así como la preparación por completo de su sistema microprocesador. Continúa con la explicación del algoritmo implementado en el mismo que permitirá dado un origen y un destino, localizar ,siempre que exista, un camino entre ambos. Se trata de un proyecto completo debido a que se ha tratado el problema a resolver desde un punto de vista teórico y se ha implementado para su resolución en la realidad. Ésto último ha permitido conocer y resolver los problemas que pueden surgir en un proyecto que salta de la teoría a la práctica.

Abstract

This document has been divided into two parts, the theoretical and practical one. First, an introduction about robotic field and its applications is presented. It also presents the target optimization problem from a transport optimization point of view. It describes the common types of graph and different types of algorithms to solve transportation problems based on graph theory.

The second part deals with the assembly of a robot vehicle as well as the complete preparation of its microprocessor system. Furthermore, it describes the proposed algorithm. The dissertation presents both the theory and the implementations details to carry out the path planning of a mobile robot. Therefore, this dissertation is a clear example of how theoretical algorithms can be put on practice successfully.

Índice

1 OBJETIVOS Y MOTIVACIÓN DEL PROYECTO

Este es un proyecto que pretende realizar un prototipo de procesado de datos y PathPlanning . En concreto se aborda el montaje y programación de un vehículo robot mediante el uso de una Raspberry Pi, mientras que otro sistema se ocupará de conocer la situación del vehículo robot desde un medio determinado y actuar modificando su ubicación en base a un determinado PathPlanning.

En particular en este proyecto se pretende el montaje y programación de un vehículo robot capaz de dar a conocer su posición global y desplazarse al lugar que se le indique. Se estudiarán los problemas que se presenten, y se dará una solución razonada y de forma eficiente.

El objetivo del proyecto es desplegar un sistema, conformado por vehículos robots móviles, que informarán de su posición global. En consecuencia, recibirá órdenes modificando según interese su ubicación. La realización de este sistema permite, entre otras cosas, conseguir detectar las condiciones terrestres de zonas que sean complicadas de explorar por los seres humanos ya sea por las condiciones del terreno o debido a que la región de exploración se encuentre, por ejemplo, en otro planeta.

Los problemas de comunicación entre el sistema que da órdenes y los vehículos robots son objetos de otro proyecto. En éste se tratará de montar, programar (mediante el uso del lenguaje de programación Python) e implementar el módulo GPS en el vehículo robot para conseguir controlar remotamente su ubicación.

La robótica es una ciencia que ha evolucionado de una forma progresiva en estas últimas décadas. Abordar un trabajo de este tipo tiene gran relevancia debido a la influencia que están teniendo los robots en todas sus formas (desde los robots fijos a los robots móviles). Con este proyecto se pretende demostrar la capacidad de un vehículo robot móvil, así como el potencial de la Raspberry Pi siendo uno de los principales objetos que se profundice y se avance en la robótica móvil.

2 INTRODUCCIÓN: ROBÓTICA,VEHÍCULOS MÓVILES Y APLICACIONES

Desde sus comienzos hasta la actualidad, la robótica ha sufrido grandes cambios. Un robot solo tiene sentido cuando su intención es la de sustituir a un trabajador de una labor desagradable o demasiado precisa. Por otra parte, al contrario de lo que se piensa, los robots no suelen ser más rápidos que los seres humanos en la mayoría de las aplicaciones, solo que al contrario que estos últimos, los robots pueden mantener su velocidad en periodos más largos.

2.1.Definición de Robot

Según la Organización Internacional para la Estandarización (ISO), el robot se define como un manipulador multifuncional reprogramable, capaz de realizar movimiento de piezas, herramientas u otros dispositivos, mediante la actuación de movimientos programados.

Otra definición disponible proporcionada por [2] es la siguiente: "Un robot es una entidad virtual o mecánica artificial. En la práctica, se trata de un sistema electromecánico que habitualmente es ejecutado por un programa de computadora o circuito eléctrico". En la Ilustración 1,se encuentra un robot con aspecto humano, siendo uno de sus principales objetivos realizar labores de rescate.

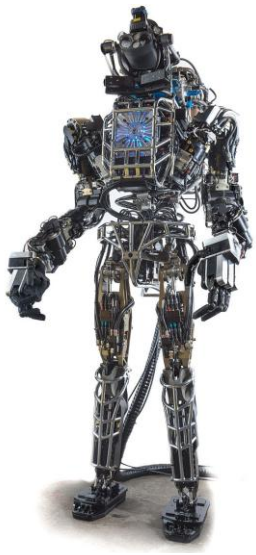

Ilustración 1: Robot Estadounidense Atlas

2.2.Introducción sobre robótica

Según [2] la robótica es la rama de la ingeniería eléctrica, mecatrónica, electrónica, mecánica, biomédica y de las ciencias de la computación que se ocupa del diseño, construcción, operación, disposición estructural, manufactura y aplicación de los robots.

Otra definición de robótica dada por [6] es la siguiente: "Técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales".

La robótica combina fundamentalmente varias disciplinas como: la mecánica, informática, inteligencia artificial, ingeniería de control, electrónica y la física. También destacan el álgebra, los autómatas programables y las máquinas de estados.

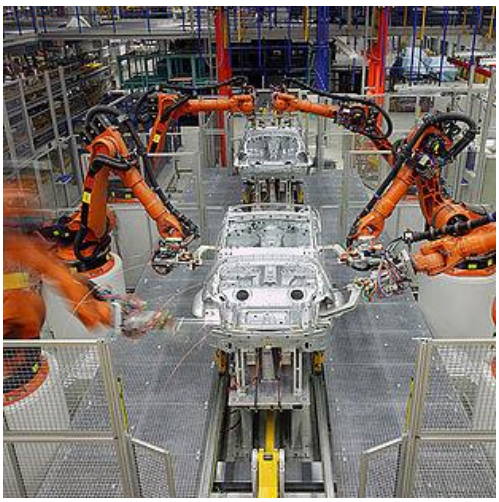

Ilustración 2 :Robótica Industrial

2.2.1 Las Leyes de la robótica

En sus comienzos la robótica fue bien vista debido a que se veía a los robots como una forma de conseguir los objetivos de las personas simplemente dándole órdenes. Es muy importante que las órdenes que reciban los robots pasen por un sistema que realice una consideración de dicha ordén categorizándola y ejecutándola siempre que su realización no afecte ni vaya en contra de la sociedad. Por ello, durante el desarrollo de la robótica se vió necesario la elaboración de unas leyes que de una forma u otra deben implantarse en los robots. Estas leyes son las siguientes:

1. Un robot no debe dañar a un ser humano ni por su ineficacia, dejar que un ser humano sufra daño.
2. Un robot debe obedecer las órdenes que le son dadas por un ser humano, excepto si dichas leyes entran en conflicto con la primera ley.
3. Un robot debe proteger su existencia, a menos que ésta entre en conflicto con las dos primeras leyes.

Sin embargo, conforme se avanzaba en otras áreas, esta ciencia ha sido marcada por aspectos que han limitado su desarrollo debido fundamentalmente al respeto de la sociedad hacia esta revolución tecnológica. Una parte de la población ha visto el mundo de los robots como una amenaza en el mundo laboral debido a que se piensa que conforme se automatizan las industrias o, más bien, conforme la tecnología robótica avanza habrá menos puestos laborales que podrán desempeñar las personas ya que estarán siendo ocupados por robots o sistemas automatizados.

Por estos motivos se llegó a elaborar una ley por parte de Fuller en 1999 que trataba este último aspecto:

4. Un robot podrá tomar el trabajo de un ser humano, pero sin dejar a esta persona sin empleo.

2.2.2 Clasificación general

De forma habitual, los robots son clasificados como industriales o de usos específicos. El objetivo de los primeros es el de realizar labores de mano de obra no cualificada o semicualificada como por ejemplo la soldadura. Por su parte, los robots de usos específicos son empleados en otros cometidos como, por ejemplo, la realización de fases de montado de un vehículo.

Dentro de los robots industriales la "Federación Internacional de Robótica" distingue cuatro tipos de robots:

- Robot de trayectoria controlable.
- Robot telemanipulado.
- Robot secuencial.
- Robot adaptativo.

Con respecto a los robots industriales de usos específicos (también conocidos como robots de servicio) pueden distinguirse en:

- Vehículos guiados automáticamente (AGV).
- Robots caminantes.
- Robots paralelos.

Para realizar una clasificación más restrictiva es necesario atender la evolución de la robótica desde los puntos de vistas cronológico y estructural.

De forma resumida, según la cronología podemos atender la Tabla 1.

Generación	**Nombre**	**Tipo de control**	**Grado de movilidad**	**Usosfrecuentes**
1ª (1982)	Pick & place	Fines de carrera, aprendizaje	Ninguno	Manipulación, servicio de máquinas
2ª (1984)	Servo	Servocontrol,trayectoria continua, progr.condicional	Desplazamientoporvía	Soldadura, Pintura
3ª (1989)	Ensamblado	Servos de precisión, visión, tacto, prog. Off-line	AGV, Guiadoporvía	Ensamblado, Desbarbado
4ª (2000)	Móvil	Sensoresinteligentes	Patas, Ruedas	Construcción, Mantenimiento
5ª (2010)	Especiales	Controlados con técnicas de Inteligencia Artificial	Andante, Saltarín	Usomilitar, Usoespacial

Tabla 1: Clasificación según T.M.Knasel

La primera generación fue caracterizada por la aparición de los robots manipuladores. Estos son sistemas multifuncionales mecánicos basados en un simple sistema de control que puede ser fijo o variable.

Dentro de la segunda generación podemos encontrar los robots de aprendizaje reconocidos como aquellos capaces de aprender una secuencia de movimientos realizadas por un agente externo como puede ser una persona y repetirla. Su forma de actuar es a través de medios mecánicos.

La tercera y cuarta generación están marcadas por la aparición del control sensorizado. Los sistemas de captación y control vienen determinados por sistemas basados en microprocesador como puede ser una computadora capaz de ejecutar un programa y enviar las órdenes convenientes al sistema robot.

En las últimas generaciones aparecen los robots inteligentes cuya principal diferencia es que poseen una gran cantidad de sensores que captan y envian sus resultados a un ordenador que ejecutará en tiempo real la toma de decisiones así como el control del sistema.

Con respecto a la estructura y basándonos en los resultados obtenidos por [2], ésta es definida en general dependiendo de la configuración del robot pudiendo ser metamórfica. Este concepto se ha introducido para aumentar la flexibilidad funcional del robot a través de cambios de configuración realizados por el mismo.

El metamorfismo puede afectar de forma normal realizando cambios elementales hasta cambios complejos como puede ser la modificación o alteración de uno o varios elementos e incluso sistemas funcionales y estructurales. Si nos basamos en la arquitectura de los robots podemos realizar una clasificación atendiendo a los siguientes grupos: móviles, poliarticulados, androides, zoomórficos e híbridos.

- Móviles: este tipo de robots poseen una alta capacidad de movimiento y desplazamiento.Estan basados por un sistema de movimiento rodante. Su movimiento puede basarse por telecontrol o por órdenes realimentadas en base a la información obtenida por sus sensores. Un ejemplo típico son aquellos robots que se dedican al transporte de piezas desde una zona a otra en una cadena de fabricación.
- Poliarticulados: se tratan de robots con diversos tipos de formas y configuración, cuya característica principal es la de ser sedentarios. Están estructurados para mover sus elementos terminales según varios sistemas de coordenadas y con un número limitado de grados de libertad. Podemos encontrar en este grupo los robots manipuladores, industriales, cartesianos entre otros.
- Androides: en este caso los robots tratan de imitar lo más fielmente posible al comportamiento cinemático humano. En la actualidad, están en desarrollo; uno de sus principales inconvenientes es conseguir la locomoción bípeda ya que es muy complejo controlar la dinámica en tiempo real y mantener su equilibrio.
- Zoomórficos: constituyen aquella familia de robots cuyos componentes y sistemas que lo componen hacen que sus movimientos y su forma de actuar se parezca al de los seres vivos. Dentro de esta clase podemos encontrar dos tipos: caminadores y no caminadores. Los tipos de robots caminantes son capaces de evolucionar para tratar superficies complejas. En cualquier caso, estos tipos de robots se usan en aplicaciones complejas relacionadas con la exploración espacial entre otras.
- Híbridos: dentro de este tipo de robots se encuentran aquellos que por su constitución cumplen con las características de varios de los anteriores.

Un esquema que permite comprender los subsistemas que se pueden encontrar en cualquiera de los tipos de robots anteriores es el siguiente[10]:

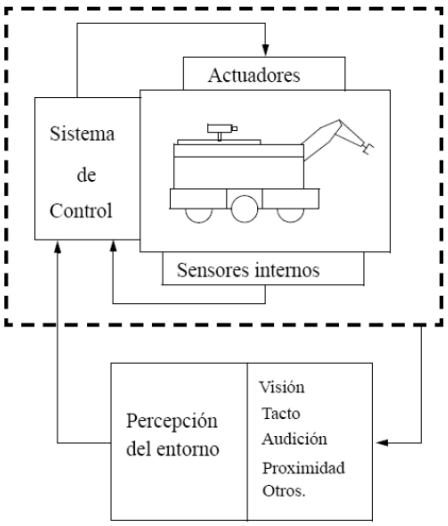

Ilustración 3: Esquema general de un sistema robot

En el esquema anterior podemos encontrar los elementos básicos que localizamos en un sistema de control convencional, siendo los siguientes:

-Entrada: en un sistema de control, la entrada se corresponde con una señal que puede ser constante (como por ejemplo un escalón) o una función variable con el tiempo (como puede ser una señal sinusoidal). En el caso del esquema anterior el sistema de control recibe como entradas los sensores internos y los externos (Percepción del entorno).

-Sistema de control o controlador: en un sistema de control, el controlador tiene un papel fundamental ya que es el encargado de recibida una entrada, establecer una salida (generalmente en forma de actuación) de tal forma que se consiga un determinado cometido. El controlador es diseñado en base a unas especificaciones y dentro de estas encontramos factores importantes como son los tiempos de subida, establecimiento y error en régimen permanente en caso de usar por ejemplo un controlador PID u otros factores en caso de usar otro tipo de control como puede ser el control predictivo.

-Salida: normalmente se tratan de sistemas de actuación sobre los que el controlador actúa para conseguir un determinado objetivo como por ejemplo la activación de unos motores.

2.3. Sensores

Los sensores son objetos que tienen la capacidad de obtener magnitudes físicas o químicas y transformarlas en variables eléctricas para su procesado. Por ejemplo se pueden encontrar sensores de temperatura, presión, intensidad lumínica, de pH, de aceleración, etc.

La principal característica de un sensor es que se encuentra en continuo contacto con la magnitud de la que obtiene la medida. Pueden estar conectados a un sistema microprocesador que permita gestionar los datos obtenidos y realizar las actuaciones oportunas.

2.3.1 Criterio de selección de sensores

La selección de un determinado tipo u otro de sensor depende en gran parte de las especificaciones del trabajo que se considere pero es aún más importante tener en cuenta la labor que realizará el sensor en cuestión y los resultados que esperemos obtener de éste. No obstante, es fundamental tener en cuenta las siguientes características propias de un sistema de instrumentación electrónica:

1. Rango: se trata de una medida de la diferencia entre los valores máximos y mínimos medidos.
2. Sensibilidad: se define como aquella proporción de cambio que se produce a la salida cuando se produce un cambio en la entrada. Otra forma de entender dicho parámetro es que sirve para localizar el cambio mínimo que tenemos que proporcionar a la entrada que permitirá observar un cambio a la salida. Es un parámetro muy importante en la linealidad y precisión.
3. Linealidad: el objetivo de este parámetro es que la representación de la entrada frente a la salida se parezca lo máximo posible a una línea recta. En forma de ecuaciónpodemosentenderlacomo:

$$y = bx$$

4. Tiempo de respuesta: es aquel que se mide desde que se produce un cambio en la entrada del sensor hasta el momento en que se aprecia un cambio en la salida fijado en un rango específico.
5. Precisión: se trata de la diferencia entre los valores reales frente a los medidos. Este parámetro debe tener especificado bajo que condiciones ha sido medido ya que puede variar dependiendo de éstas.
6. Repetibilidad: se conoce como la forma de medir la diferencia entre varias mediciones consecutivas y bajo las mismas condiciones. Con este parámetro es posible estimar que valor se obtendrá en las sucesivas medidas.

7. Resolución: es la menor diferencia de indicación de un dispositivo visualizador que puede percibirse de forma significativa. También es usado para indicar el mínimo incremento que se puede dar en una medida.
8. Tamaño y peso: este parámetro es especialmente delicado y depende estrictamente de la aplicación que se vaya a dar al sensor.
9. Interconexión: el tratamiento de este factor de los sensores es útil ya que puede ser determinante para decidir sobre un sensor u otro ya que en muchos casos es necesario disponer de sensores que permitan su interconexión con otros sensores .
10. Confiabilidad: de igual forma que el tamaño y peso, este parámetro depende de la aplicación del sensor y en función de dicha aplicación puede decirse que es el parámetro más importante debido a que nos indica las condiciones óptimas del sensor y su porcentaje de fallo (como puede ser el tiempo medio sin averías). Es evidente que este parámetro en la mayoría de los casos debe tener un porcentaje lo más elevado posible para que el funcionamiento del sensor y ,por tanto, del sistema que se sirva sea el más óptimo y eficiente.

2.4. Robots móviles

Los robots móviles son máquinas que tienen la capacidad de desplazarse de un lugar a otro de forma telecontrolada o autónoma con el objeto de realizar una tarea concreta[1]. Sin embargo, los robots móviles actuales no están adaptados al entorno doméstico de los humanos. En este sentido, han surgido en los últimos años gran cantidad de proyectos encaminados a dotar a los robots de un cierto grado de humanidad, con una morfología similar a la humana y con el objetivo de hacerlos trabajar como robots de servicio en nuestras oficinas o viviendas.Un claro ejemplo lo podemos localizar en la Ilustración 4 donde podemos apreciar un AGV realizando labores de logística.

Ilustración 4: Vehículo de guiado automático dedicado a Logística

Uno de los campos de aplicación más interesantes de la robótica móvil consiste en robots capaces de operar en condiciones exteriores sobre terrenos no preparados (robots planetarios, robots en agricultura, robot en operaciones de búsqueda y rescate, robots militares, etc.). Sin embargo, conseguir que los robots se muevan de forma eficiente y con gran precisión en este tipo de entornos no es sencillo. En la Ilustración 5, encontramos al vehículo "RoverCuriosity" cuyas aportaciones están siendo muy relevantes en la exploración del planeta Marte.

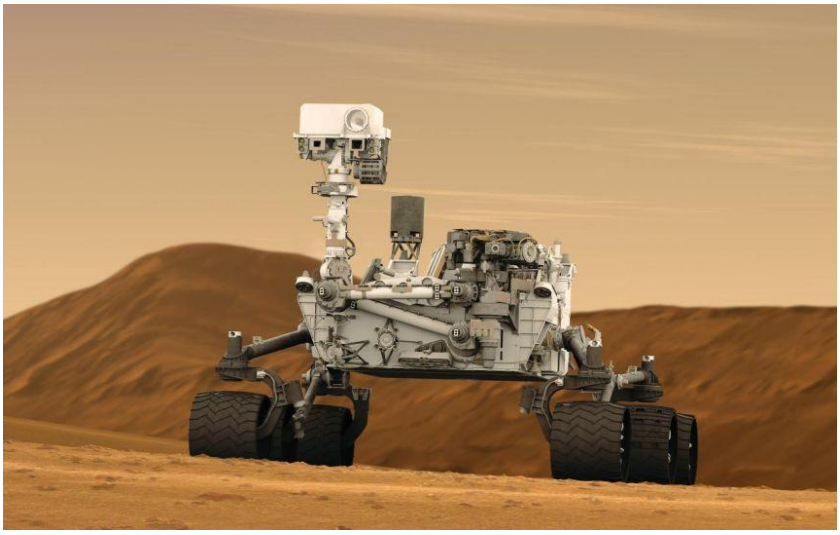

Ilustración 5: RoverCuriosity

Una clasificación de robots móviles es la siguiente:

- **Robots con ruedas:** Son el tipo de robot más común, siendo utilizado principalmente en el transporte de mercancías. Las ruedas proporcionan a este tipo de robot una movilidad con gran eficiencia (relación entre la distancia recorrida y la energía consumida). Su uso queda restringido habitualmente a superficies planas con pocas irregularidades. En [4] hay disponible más información.

Ilustración 6: Robot Movil con ruedas

- **Robots orugas o con cadena:** Estos tipos de robots no son simples robots con ruedas. Su peculiar estructura los hacen ser los elegidos para moverse con gran eficiencia (aunque menor que la de los robots con ruedas) por terrenos irregulares.En [5]se tiene disponible más información.

Ilustración 7: Robot oruga o con cadena

- **Robots con patas o zoomórficos:** En este caso la morfología se asemeja a la de animales terrestres con patas (como por ejemplo las arañas). A pesar de que tienen una baja eficiencia en su movilidad, son capaces de adaptarse con gran precisión a los terrenos irregulares.En [3] puede encontrar más detalles.

Ilustración 8: Robot con patas o zoomórficos

- **Robots aéreos:** Son robots capaces de realizar vuelos sin tripulación (conocidos como Unmanned Arial Vehicles) o mediante control remoto (Remotely Piloted Vehicle). Su uso está restringido actualmente a aplicaciones militares, de observación y vigilancia ya que son capaces de mantener un gran nivel de control en lo referente al vuelo. Su aspecto es parecido al de los vehículos aéreos tripulados tradicionales como son el avión y el helicóptero aunque su tamaño suele ser inferior al de estos ya que presentan limitaciones respecto al hardware y a los sensores que se pueden implementar en ellos. Dependiendo de la aplicación en la que se usen presentan distinto grado de autonomía.

Ilustración 9: UAV MQ-9 Reaper realizando labores militares

- **Robótica espacial:** se tratan de aquellos robots usados fuera de la atmósfera terrestre. El diseño de estos robots viene condicionado fundamentalmente por la necesidad de vencer la gravedad terrestre y por tanto cuenta con grandes restricciones en peso y volumen. En estos robots la fiabilidad juega un papel fundamental debido a la gran dificultad que hay cuando se tienen que realizar reparaciones. En la Ilustración 10, podemos apreciar el "RoverLunokhod 1", quién realizó una importante expedición en la luna, siendo un claro ejemplo de que la robótica móvil es fundamental para el avance en las investigaciones relacionadas con el espacio debido a que se puede considerar como una de las misiones de exploración con robots móviles con más éxito de la historia.

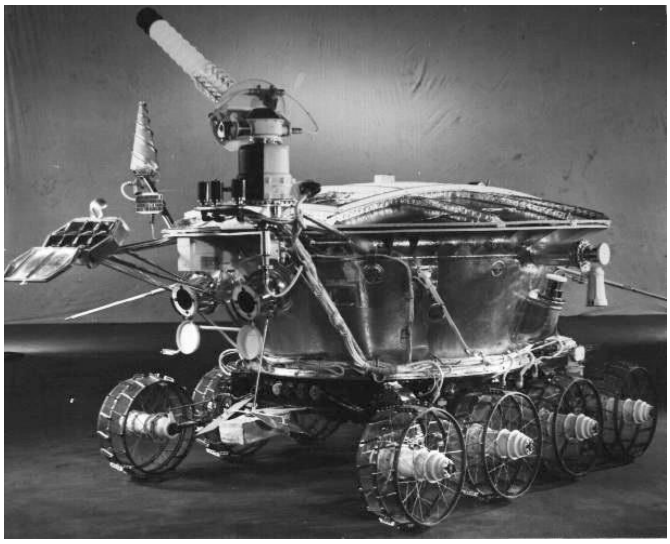

Ilustración 10: Rover Lunokhod 1

2.5. Sistemas microprocesadores para robots

La interfaz que permite la comunicación y ejecución de tareas entre un humano y un robot es habitualmente un sistema microprocesador como es una computadora. Mediante el uso de ordenadores para tratar los robots es posible aprovechar al máximo las prestaciones del mismo mediante el uso de lenguajes de programación con los que se implementan diversos algoritmos en funcion de las aplicaciones del sistema robot.

El avance de los sistemas computacionales es esencial para que los robots aumenten sus capacidades y prestaciones. La principal tarea de un computador que controle un sistema robot es que debe cumplir con la ejecución de tareas con restricciones de tiempo involucrando para ello la evaluación de cálculos complejos como pueden ser procesado de grandes cantidades de datos, multiplicaciones o tratamiento de matrices e incluso evaluación de funciones matemáticas como pueden ser las funciones trigonométricas.

2.5.1 Eficiencia Software

Para conseguir cumplir con los tiempos de ejecución es muy importante que los algoritmos implementados por software sean lo más eficientes posible. Para conseguirlo es aconsejable realizar entre otras cosas:

- **Uso de aritmética basada en numeros enteros:** para conseguir un algoritmo eficiente se deben evitar la representación de números basados en puntos flotantes, ya que provocan un gran consumo en su tratamiento. Para evitarlos, se usa la técnica conocida como escalamiento que consiste en representar el número flotante como entero e introducirle un factor de escalamiento. Por ejemplo, si consideramos el número flotante 5.75, éste se representará como 575 y estará multiplicado por un factor de escalamiento de 1/100.

  Este aspecto se realiza debido a que se sabe que la aritmética de números enteros es habitualmente de 5 a 10 veces más rápida con respeto a la flotante. Aunque el uso de este método es eficiente se debe tener en cuenta la pérdida de precision que el factor de escalamiento puede introducir y que ,por tanto, puede afectar al funcionamiento del robot.

- **Tratamiento de funciones trigonométricas:** para obtener resultados eficientes en algoritmos que se sirvan de resultados obtenidos de la aplicación de funciones trigonométricas es importante evitar el uso de dichas funciones siempre que sea posible,ya que pueden involucrar cálculos con números con representación flotante. En su lugar, se usan sistemas de almacenamiento como son las tablas donde se recogen en cada posición los valores que se obtendrían al ejecutar la función que corresponda. Se sabe que el almacenamiento de dichos valores en una tabla ocupan menos memoria que la que necesitará el programa para obtener los valores mediante el uso de funciones y además la obtención del resultado es prácticamente inmediata ya que no es necesario calcularlo sino obtenerlo a partir de la tabla. Si fuera necesario más precisión que la proporcionada por la tabla pueden usarse otras técnicas como la interpolación.

En [1] se encuentran más detalles y métodos a tener en cuenta para el desarrollo eficiente del software.

2.5.2 Consideraciones Hardware

El hardware que utilice el sistema robot debe tener unos requerimientos que doten al sistema de las funciones que requiera y cumpla con los tiempos de ejecución. Para ello, es necesario tener en cuenta que existan sistemas que se encarguen de los cometidos siguientes:

- Sistema de obtención de datos sobre el entorno donde se encuentra el robot, es decir, sensorización externa.
- Sistema de obtención de datos sobre el estado interno del robot, es decir, sensorización interna.
- Capacidad (depende del robot) de realizar labores de manipulación de objetos.
- Sistema que permita modificar las actuaciones del robot de forma programada en base a los datos (internos y externos) obtenidos.

En base a la información recogida por [9] para conseguir un funcionamiento hardware eficiente se debe atender al uso de una arquitectura funcional basada en tres modelos funcionales donde el primero se encargaría de la gestión de la información obtenida por los sensores, el segundo por su parte trataría de asignar la información más importante del entorno y el tercero se encargaría de aplicar los resultados de los módulos anteriores y tomar la decisión más adecuada en base a los mismos.

Conseguir un funcionamiento hardware que cumpla con las especificaciones del sistema robot no es una labor sencilla, ya que hay que tener en cuenta diversos factores como son las tareas que debe realizar el robot, cada cuanto tiempo deben realizarse, el consumo de los componentes que lo constituyen, la temperatura y humedad máxima capaz de soportar, por ello se disponen en la actualidad de una gran cantidad de sistemas microprocesadores en base a cada una de las necesidades que abarcan desde las necesidades más básicas a las más complejas.

Con la evolución tecnológica de los sistemas basados en microprocesador han aparecido placas capaces de comportarse como un computador de prestaciones básicas a un precio muy asequible, ideal para realizar pruebas a bajo coste como es el caso de los sistemas basados en Arduino o la placa Raspberry Pi cuya aportación ha sido fundamental para el desarrollo de este trabajo.

2.5.3 Sistemas Operativos de robots

Un sistema operativo es el software principal o conjunto de programas de un sistema informático que gestiona los recursos de hardware y provee servicios a los programas de aplicación de software, ejecutándose en modo privilegiado respecto a los restantes[2].

Ilustración 11: Situación del Sistema Operativo

En la actualidad, existen una enorme cantidad de sistemas operativos como son Microsoft Windows, Unix, Ubuntu Linux, Debian, OS X. Referente a la robótica el sistema operativo que tiene un gran impacto es ROS (Robot Operating System).

ROS es un framework programado en C++ y Python para el desarrollo de software para robots que proporciona funciones de un sistema operativo en un cluster heterogéneo. Con el uso de este sistema operativo, el usuario podrá disponer de servicios como: abstracción del hardware, control de dispositivos de bajo nivel, implementación de funcionalidad de uso común, paso de mensajes entre procesos, localización , simulación, etc. Su uso queda adaptado en la actualidad a Ubuntu GNU/Linux.

No obstante, en la actualidad existen sistemas operativos basados en GNU/Linux que también sirven como medio de programación e intermediario entre el usuario y el robot como es el caso de Raspbian, sistema operativo que ha permitido la resolución de este trabajo.

2.5.4 Programación de robots

Una de las principales ventajas que presentan los robots frente a otros sistemas es la capacidad de reprogramación para adecuarse a nuevos cambios (siempre que sus prestaciones lo permitan). Para poder servirse de esta cualidad, es necesario contar con una programación del sistema eficiente que permita comunicarse con el robot en el menor tiempo posible para transmitirle aquellas tareas que debe ejecutar.

En lo que respecta a los robots industriales, hay disponible dos modos de programación: online y offline.Mientras que la programación online se caracteriza por realizarse en el mismo lugar donde se encuentra el robot , la offline se lleva a cabo en un ordenador permitiendo realizar simulaciones.

La programación de los robots se lleva a cabo mediante el uso de lenguajes de programación. Hay una gran cantidad de lenguajes de programación de robots (basados en su mayoría en el lenguaje ensamblador) entre los que se encuentran: ANORAD, EMILY,RCL,RPL,SIGLA etc.

En lo que respecta a la robótica convencional actual, el avance de los sistemas de computación ha permitido el desarrollo de diversos lenguajes de programación entre los que podemos destacar C/C++, Java y Python. Estos lenguajes, a diferencia de los anteriores son conocidos como lenguajes de programación de alto nivel ya que constan de librerías con funciones y métodos, variables y otros parámetros característicos de cada lenguaje que permiten al usuario subir a unos niveles de abstracción de tal forma que la labor de programación es más sencilla, ya que evita realizar tareas laboriosas propias de los lenguajes de bajo nivel, como por ejemplo aquellos basados en ensamblador.

En concreto, la realización de este trabajo ha sido posible mediante la implementación de algoritmos basados en el lenguaje de programación Python.

2.6.Aplicaciones generales de la robótica

En la siguiente lista se encuentra una clasificación que ha sido realizada por la "Federación Internacional de la Robótica" y engloba una gran cantidad de tareas empleadas por los robots en la actualidad.

- Manipulación en fundición.
- Manipulación en moldeo de materias plásticas.
- Manipulación en la forja y estampación.
- Soldadura: al arco, por puntos y por láser.
- Aplicación de materiales como la pintura, adhesivos y secantes.
- Mecanización.
- Labores de montaje.
- Paletización.
- Medición, inspección, control de calidad.
- Manipulación de materiales.
- Formación, enseñanza e investigación.
- Otros.

Cabe destacar que si bien el listado anterior contempla una gran cantidad de aplicaciones, en la actualidad la robótica se ha expandido a más campos como son la agricultura, la construcción, usos domésticos, espacio, medicina, vigilancia y seguridad entre otros.

Ilustración 12: Aspirador de uso doméstico iRobot Roomba 580

3 PATH PLANNING

La planificación de rutas o path planning es un concepto relacionado con el enrutamiento de sistemas como pueden ser aquellos donde se realicen transmisiones de paquetes en redes de telecomunicaciones o el uso de métodos para obtener un determinado objetivo como la obtención de la ruta mínima entre un determinado origen y un destino.

En los sucesivos apartados, se van a presentar distintos problemas que involucran al path planning y sus algoritmos de resolución.

3.1. Planificación de rutas: problema de optimización

Para la obtención de una ruta con unas especificaciones concretas es necesario imponer una serie de condiciones como pueden ser la obtención del camino más corto o más largo en una determinada región. Se trata, por tanto, de resolver un problema de optimización de un sistema que se puede modelar matemáticamente y cuya resolución es factible usando los métodos propios de la optimización.

El problema de optimización puede definirse como: **dado un conjunto X y una determinada función que asigna a cada x (perteneciente al conjunto X) un valor numérico f(x), se desea para el caso mínimo encontrar un x_0 que pertenezca a X y que cumpla la condición [11]**:

$$f(x_0) \leq f(x) \text{ para todo x de X}$$

Para el caso máximo es completamente dual al anterior obteniendo como resultado:

$$f(x) \leq f(x_0) \text{ para todo x de X}$$

Para los siguientes apartados se tratarán los casos máximos y mínimos de forma abreviada usando la siguiente notación:

$$f(x_0) = Maxf(x) \text{ para el caso máximo.}$$

$$f(x_0) = Minf(x) \text{ para el caso mínimo.}$$

Donde $f(x)$ es conocida como función objetivo y puede tomar el valor de un determinado recurso del sistema en función del problema que se esté resolviendo por ejemplo: coste, tiempo, productos.

La obtención del conjunto X determinará las restricciones del sistema a resolver. El problema de optimización puede tener una solución , o no debido fundamentalmente a que no es posible alcanzar un valor máximo o mínimo en la función objetivo.

Es muy importante tener en cuenta las siguientes propiedades asociadas:

- $Max(f(x) + a) = Maxf(x) + a$, donde a es una constante.
- $Max\big(cf(x)\big) = cMaxf(x)$, donde c es una constante positiva.
- $Min\big(bf(x)\big) = bMaxf(x)$, donde b es una constante negativa.
- $Minf(x) = -Max(-f(x))$.
- $Maxf(x) = -Min\,(-f(x))$.

3.1.1 Programación Matemática

La resolución de problemas de optimización viene especialmente condicionada por este método ya que define los pasos necesarios para abordar dichos problemas.

De forma general, un problema de optimización puede representarse de la siguiente forma:

Maximizar (minimizar) $z = f(x_1, x_2, \dots x_n)$ sujeto a las restricciones:

$$q_1(x_1, x_2, \dots x_n)[\leq, =, \geq]b_1$$
$$q_2(x_1, x_2, \dots x_n)[\leq, =, \geq]b_2$$
$$q_m(x_1, x_2, \dots x_n)[\leq, =, \geq]b_m$$

donde $f(x_1, x_2, \dots x_n), q_1(x_1, x_2, \dots x_n), \dots, q_m(x_1, x_2, \dots x_n)$ son funciones que dependen de n variables y $b_1, \dots, b_m$ son constantes. Cada restricción emplea uno de los signos $[\leq, =, \geq]$.

El conjunto X que define el problema que se esté abordando está compuesto por aquellos $x = (x_1, x_2, \dots x_n)$ que cumplan con todas las condiciones obteniendo de este modo lo conocido como soluciones admisibles o básicas ya que se encuentran dentro de la región de admisibilidad.

La resolución de un problema de optimización concreto consta de pasos sistemáticos cuando aplicamos los métodos de la programación matemática, en concreto se realiza lo siguiente:

- Identificación del problema a resolver.
- Determinación de aquella variable que se desea optimizar y ,con ello, las variables de entrada $x_1, x_2, \dots x_n$ consiguiendo expresar la primera en función de las últimas.
- Determinación de las restricciones del problema a abordar y establecerlo de forma matemática.
- Aquellas variables que ,por ejemplo, sólo puedan tomar valores enteros o solo puedan tomar valores mayores que un determinado número u otras condiciones deben indicarse.

Tras realizar estos pasos se encuentra un problema modelado donde es posible proceder a su resolución mediante técnicas de optimización con el objetivo de encontrar el valor óptimo y ,con ello, una solución óptima siempre que el modelo matemático lo permita.

3.1.2 Programación Lineal

Los problemas de programación lineal presentan la siguiente forma:

$$\text{Maximizar (Minimizar)} \; z = c_1x_1 + c_2x_2 + c_nx_n$$
$$\text{sujeto a : } a_{11}x_1 + a_{12}x_2 + a_{1n}x_n \; [\leq, =, \geq] \; b_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{2n}x_n \; [\leq, =, \geq] \; b_2$$
$$a_{j1}x_1 + a_{j2}x_2 + a_{jn}x_n \; [\leq, =, \geq] \; b_j$$
$$a_{m1}x_1 + a_{m2}x_2 + a_{mn}x_n \; [\leq, =, \geq] b_m$$

,donde $c_1, c_2, \dots c_n$ son unos coeficientes conocidos como costes, $a_{11}, a_{j1}, \dots a_{mn}$ (datos técnicos) y $b_1, b_2, \dots b_m$ (coeficientes de restricciones) son constantes.

Se está suponiendo que todas las funciones presentes tienen un comportamiento lineal. La solución admisible será aquella que cumpla con todas las restricciones. El óptimo viene determinado en base a si la función objetivo trata de maximizar o minimizar y será en su caso aquel valor que cumpla con todas las restricciones y sea el máximo z_{max} o mínimo z_{min} .

Los resultados de la programación lineal son los siguientes:

1. La región admisible es un polígono, es decir, una región del plano delimitada por rectas.
2. Si la función objetivo presenta óptimo, estará localizado en uno de los vértices del polígono.

Conclusión determinante: **si se cumplen los resultados anteriores y el polígono es cerrado (sus lados forman una poligonal cerrada) la función objetivo siempre tiene óptimo.**

Importancia de la forma estándar

El objetivo de la programación lineal es resolver un determinado problema para alcanzar su óptimo. Para ello, ésta se sirve del algoritmo de resolución conocido como "método símplex".

Para alcanzar el óptimo mediante el uso de dicho método, es necesario que se cumpla con la forma estándar ya que está diseñado en base a ésta. Lo más habitual es que los problemas de programación lineal no vengan dado por su forma estándar así que dado un problema se realiza lo siguiente:

- Si no viene determinado por su forma estándar, realizar su conversión.
- Aplicar el método símplex para conseguir el óptimo.

De forma resumida:

Se parte de P.Lineal→P.Lineal Estándar → Aplicación Símplex→ Óptimo.

Técnicas de conversión a la forma estándar

Para llegar a la forma estándar deben atenderse los siguientes aspectos:

- Si el término b_i de la restricción i es negativo, se procederá a cambiarle el signo a positivo multiplicando en ambos lados por -1 realizando los cambios de signos y desigualdades que procedan.
- Todas las restricciones con signos de desigualdad deben ser reemplazadas por signos de igualdad. Para ello, se introducen lo conocido como variables de holgura h_i que tomarán signo positivo si la condición de desigualdad es ≥ y negativo si la condición es ≤.
- Todas las variables deben de tomar valores postivos. Si alguna variable puede tomar también valores negativos, será sustituida por la diferencia de dos variables no negativas (u y v). Por ejemplo si x_j puede tomar valores positivos y negativos, se realiza la conversión $x_j = u_j - v_j$, donde u y v son no negativos.
- Las variables que toman valores negativos o cero serán sustituidas por otra no negativa con signo menos. Por ejemplo si $x_j \leq 0$ se realiza la conversión $x_j = -u_j$.

Como resultado, tras aplicar la forma estándar a un problema cuyas restricciones vienen dadas por el signo ≤ nos encontramos con el problema de programación lineal en forma estándar:

$$\text{Maximizar (Minimizar)} \; z = c_1x_1 + c_2x_2 + c_nx_n$$

$$\begin{aligned} \text{sujeto a :} \; & a_{11}x_1 + a_{12}x_2 + a_{1n}x_n - h_1 = b_1 \\ & a_{21}x_1 + a_{22}x_2 + a_{2n}x_n - h_2 = b_2 \\ & a_{j1}x_1 + a_{j2}x_2 + a_{jn}x_n - h_j = b_j \\ & a_{m1}x_1 + a_{m2}x_2 + a_{mn}x_n - h_m = b_m \\ & a_{ij} \geq 0, h_i \geq 0 \end{aligned}$$

El siguiente paso es la aplicación del algoritmo de resolución de problemas de programación lineal símplex que se tratará como algoritmo de path planning.Para más detalles consulte[11] y [12].

3.2. Introducción al problema de transporte

El problema de transporte o de redes de distribución se trata de un tratamiento especial en programación lineal de los problemas relacionados con la necesidad de llevar determinadas unidades desde un lugar determinado denominado Fuente hasta un determinado lugar denominado Destino.El objetivo fundamental del problema de transporte es realizar con éxito así como satisfascer los requerimientos dados por los destinos con un coste mínimo. En términos generales, el problema de transporte se describe de acuerdo a la siguiente terminología:

Sean m nodos fuentes (también denominados orígenes) que deben enviar un determinado número de unidades de un producto específico a n destinos. Cada fuente i está limitada a suministrar s_i unidades del producto y cada destino j debe recibir como máximo d_j unidades. El coste de enviar desde el nodo origen i hacia el nodo destino j es c_{ij}, de forma que si entre i y j se envian x_{ij} unidades, el coste respectivo es $c_{ij}x_{ij}$.

El objetivo es conseguir enviar todo lo demandado por los nodos destinos a un coste total mínimo, es decir conseguir las cantidades x_{ij} que deben enviarse desde el origen hasta el destino de tal forma que el coste sea mínimo, es decir , $c_{ij}\,x_{ij}$ sea mínimo.

3.2.1 Modelo general

Como hemos indicado previamente, el problema de transporte es un caso especial del problema de programación lineal. En concreto:

Sea z el coste total de una determinada distribución de unidades x_{ij}que se distribuyen entre el origen i hacia el destino j. En términos de programación lineal podemos modelarlo de la siguiente forma:

$$\text{Minimizar } z = \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\,x_{ij}$$

$$\text{sujeto a } \sum_{j=1}^{n} x_{ij} = s_i \text{ para i= 1,2,...,m.}$$

$$\sum_{i=1}^{m} x_{ij} = d_j \text{ para j= 1,2,...,n.}$$

$$x_{ij} \geq 0 \text{ , para toda i y j.}$$

Como podemos apreciar, cualquier problema de programación lineal que se ajuste a la forma anterior, es considerado como un problema de transporte independientemente del problema en cuestión. Éste es uno de los motivos fundamentales por lo que el problema de transporte es uno de los problemas más importantes de la programación lineal. En la Tabla 2, se encuentra un resumen de los parámetros más importantes.

	Destino				
Origen	1	2	…	n	Recursos
1	c_{11}	c_{12}	……	c_{1n}	s_1
2	c_{21}	c_{22}	……	c_{2n}	s_2
…	……	……	……	……	…
m	c_{m1}	c_{m2}	……	c_{mn}	s_m
Demanda	d_1	d_2	……	d_n	

Tabla 2: Resumen de parámetros de transporte

Se dice que un problema de transporte tiene soluciones admisibles si el total de unidades suministradas es igual a las demandadas, es decir, si se cumple que:

$$\sum_{i=1}^{m} s_i = \sum_{j=1}^{n} d_j$$

En este caso, se trata de un problema de transporte balanceado ya que se consigue suministrar a los destinos todo lo que éstos demandan a las fuentes.

No obstante, es posible que el problema no sea balanceado, es decir, que puede ocurrir que haya más productos a suministar que demandas, o que haya más demandas que productos a suministrar. Si implementamos esta idea en las ecuaciones del modelo lineal se obtiene un sistema de ecuaciones incompatible, indicando que no hay solución. Sin embargo, para tratar este problema se usan los siguientes métodos:

- Si hay más demanda que suministro, se agrega una fuente ficticia cuyo suministro es la diferencia restante.
- Si hay más suministro que demanda, se agrega un destino ficticio cuya demanda será la diferencia restante.

Otra consideración a tener en cuenta es que hay determinados problemas donde los suministros y la demanda solo pueden tomar valores enteros y ,por tanto, la solución también debe obtenerse de forma entera. Este aspecto obtuvo como resultado la siguiente **propiedad de soluciones enteras** que anuncia que: **en aquellos problemas donde los s_i y los d_j solo pueden tomar valores enteros, todas las variables básicas y con ellos las soluciones admisibles se obtendrán enteras.**

Para más información acerca del problema de transporte en general se recomiendan [11],[12].

3.2.2 Representación mediante grafos

Según [2] un grafo es un conjunto de objetos llamados nodos unidos por enlaces denominados arcos que permiten representar relaciones binarias entre elementos de un conjunto. Típicamente, un grafo se representa como un conjunto de puntos (nodos) unidos por líneas (arcos).

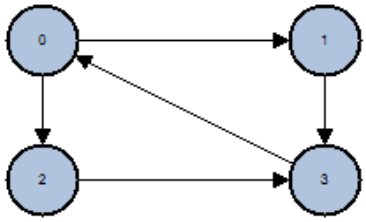

Ilustración 13: Ejemplo de grafo

Grafos Dirigidos

Un grafo dirigido consiste en un conjunto N de nodos y un conjunto A de arcos cuyos elementos son pares ordenados de distintos nodos.

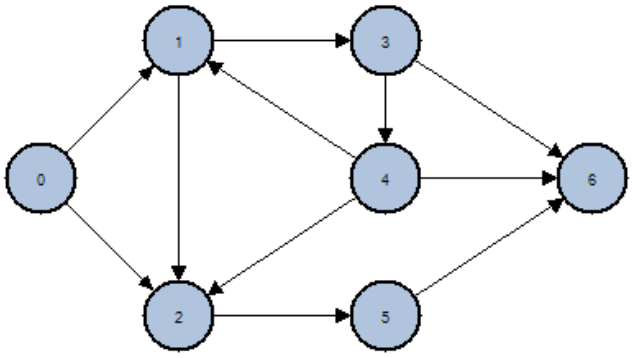

Ilustración 14: Grafo Dirigido

Red Dirigida

Una red dirigida se trata de un grafo dirigido cuyos arcos presentan asignados unos valores numéricos que se denominan costes.

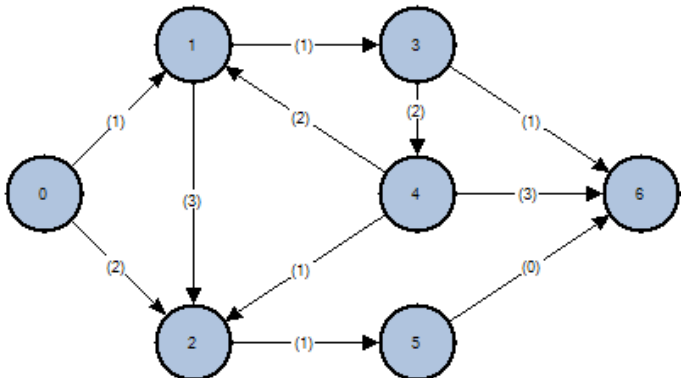

Ilustración 15: Red Dirigida

Grafos no dirigidos

Este tipo de grafo consiste en un conjunto N de nodos y un conjunto A de arcos cuyos elementos son pares no ordenados de nodos distintos. En este tipo de grafos los arcos son bidireccionales, es decir, permite el flujo en ambas direcciones desde los nodos que comprenden dicho arco.

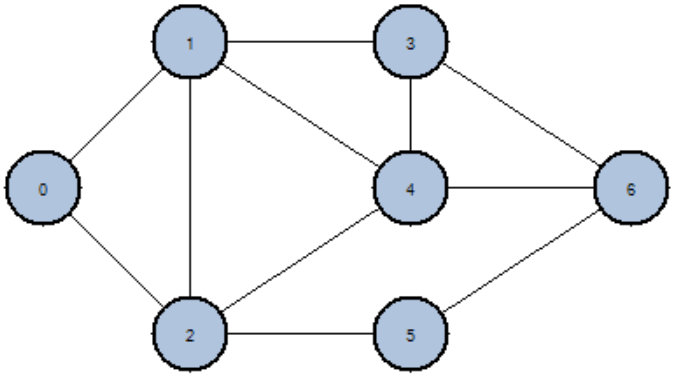

Ilustración 16: Grafo no dirigido

Conceptos importantes de grafos

- **Grados:** el grado interno de un nodo es el número de arcos que llegan al nodo y su grado externo es el número de arcos que salen del nodo. La suma de ambos es el grado del nodo. Como puede deducirse, la suma de los grados internos de todos los nodos es igual a la suma de los grados externos de todos los nodos y ambos coinciden con el número de arcos en la red.
- **Subgrafo:** Un grafo $G^{'} = (N^{'}, A')$ con $N^{'}$ siendo el número de nodos y $A^{'}$ el número de arcos, se trata de un subgrafo de $G = (N, A)$ si $N^{'}$ está contenido en N y A' está contenido en A.
- **Camino dirigido:** un camino dirigido es una versión orientada de un camino en el sentido que para dos nodos consecutivos i_k e i_{k+1} en el camino, se tiene que $(i_k, i_{k+1}) \in A$, donde A es el número de arcos de un determinado grafo.

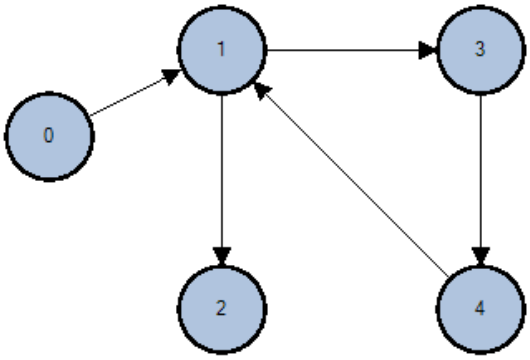

Ilustración 17: Camino Dirigido

- **Camino no dirigido:** un camino no dirigido es una versión orientada de un camino en el sentido que para dos nodos consecutivos i_k e i_{k+1} en el camino, donde no se tiene que $(i_k, i_{k+1}) \in A$, donde A es el número de arcos de un determinado grafo.

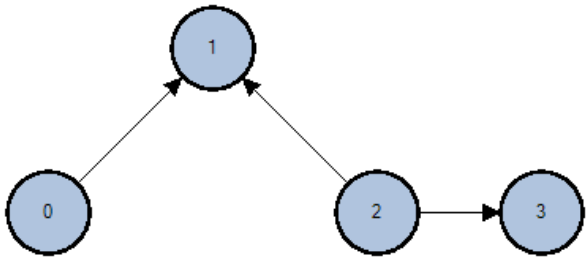

Ilustración 18: Camino No Dirigido

- **Ruta:** una ruta se trata de un camino sin repetición de nodos. En un grafo con caminos, es posible particionar los arcos de dicho camino en dos grupos: aquellos que van hacia delante y aquellos que van hacia atrás. Aquellos arcos (i, j) va dirigido hacia delante si la ruta visita al nodo i antes de visitar al nodo j, en caso contrario será un arco que está dirigido hacia atrás.
- **Ruta dirigida:** una ruta dirigida será aquella en la que no se repiten nodos en su transcurso. Visto desde otra perspectiva, en las rutas dirigidas no hay arcos dirigidos hacia atrás.
- **Ciclo:** Un ciclo es una ruta $i_1 - i_2 - \cdots i_r$ junto con el arco (i_1, i_r) o (i_r, i_1).

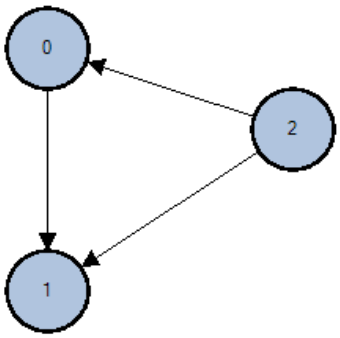

Ilustración 19: Ciclo

- **Ciclo dirigido:** un ciclo dirigido es una ruta dirigida $i_1 - i_2 - \cdots i_r$ junto con el arco (i_r, i_1)

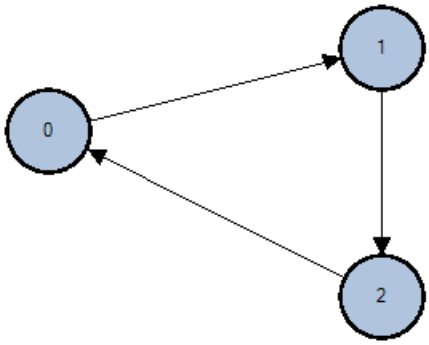

Ilustración 20: Ciclo Dirigido

- **Conectividad:** Dos nodos i, j están conectados siempre que el grafo contenga por lo menos una ruta desde el nodo i hacia el nodo j. Se dice que un grafo es conectado si cada par de nodos están conectados. De cualquier otra forma se considera que el grafo es desconectado.
- **Conectividad fuerte:** un grafo conectado se considera fuertemente conectado siempre que contenga como mínimo una ruta dirigida desde cada nodo hacia todos los demás nodos.
- **Árbol:** se entiende por árbol a un grafo conectado que no contiene ciclos. Sus propiedades son las siguientes:
    - Un árbol de n nodos contiene exactamente $n - 1$ arcos.
    - Un árbol como mínimo contiene dos nodos de grado 1.
    - Cada dos nodos de un árbol están conectados por una ruta única.

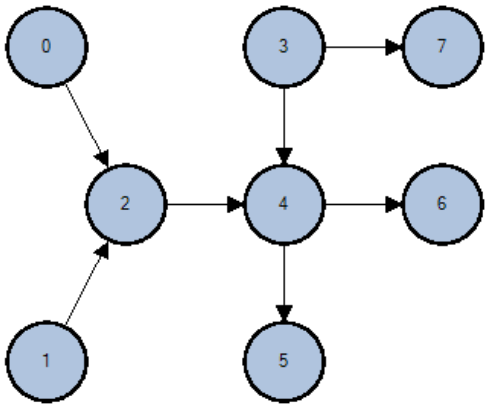

Ilustración 21: Árbol

- **Matriz de adyacencia de nodo a nodo:** consiste en la representación de la red como una matriz $H = [h_{ij}]$ de tamaño nxn. Dicha matriz tiene una fila y una columna por cada nodo siendo la i-ésima entrada $h_{ij} = 1$ si el arco $(i, j) \in A$ y cero en cualquier otro caso. Esta forma de representar es especialmente eficiente si la red es lo suficientemente densa ya que para redes esparcidas se desperdicia mucho espacio de dicha matriz. Para almacenar los costes y capacidades de los arcos se pueden usar dos matrices cuadradas independientes $C\ y\ U$ respectivamente.

  Para determinar el coste o la capacidad de un determinado arco (i, j) tan solo es necesario ubicar los elementos situados en dichas posiciones en las matrices $C\ y\ U$. Conocer los arcos que salen del nodo i consiste en observar los elementos con valor 1 en la fila i-ésima de la matriz H. De forma dual se pueden obtener aquellos arcos que llegan al nodo j sin más que examinar los elementos con valor 1 de la columna j en la matriz H.

3.3. Algoritmos de Path planning

En los sucesivos apartados se irán comentando de forma breve en que consisten determinados algoritmos útiles en aplicaciones que involucren la planificación de rutas o path planning.

3.3.1 Algoritmo Símplex

El método símplex consiste en un procedimiento que permite obtener una solución óptima en un problema de programación lineal estándar dado. Este método destaca por ser capaz de resolver modelos más complejos que los tratados de forma gráfica y además sin restricción en el número de variables a utilizar.

El objetivo de este método es conseguir mejorar la solución obtenida mediante iteración. Desde un punto de vista matemático, el algoritmo símplex consiste en ir caminando de un vértice del poliedro (que constituye la región admisible) hacia el vértice cercano de forma que en función del contexto de la función objetivo (minimizar o maximizar) se vaya mejorando la solución hasta dar con aquella que es la mejor y ,por tanto, óptimo.

Como se detalló en apartados anteriores (véase el apartado 3.1.2) para aplicar este método es necesario que el problema se encuentre en la forma estándar para partir de una solución básica admisible.

Resolución a partir de la forma tabular

El algoritmo símplex de forma tabular realiza operaciones de forma directa sobre los datos de un problema en la forma:

$$\text{Maximizar (Minimizar)} \; z = c_1x_1 + c_2x_2 + c_nx_n$$

$$\text{sujeto a :} a_{11}x_1 + a_{12}x_2 + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{2n}x_n = b_2$$

$$a_{j1}x_1 + a_{j2}x_2 + a_{jn}x_n = b_j$$

$$a_{m1}x_1 + a_{m2}x_2 + a_{mn}x_n = b_m$$

$$a_{ij} \geq 0$$

donde todas las variables son positivas, es decir $x_i \geq 0$ y partiendo de una solución básica admisible de la forma $x_1', x_2', \dots, x_m'$.

Para localizar el óptimo, el algoritmo simplex emplea la siguiente tabla:

v.bás	x_1	……	x_j	……	x_n	b
x_1'	a_{11}	……	……	……	c_{1n}	b_1
…						
x_i'	a_{i1}	……	a_{ij}	……	a_{in}	b_i
…						
x_m'	a_{m1}	……	a_{mj}	……	a_{mn}	b_m
c	c_1	……...	c_j	……...	c_n	0
c'	c_1'	……...	c_j'	……...	$c_{n'}$	$-z_0$

Tabla 3: Método Símplex en forma tabular

De forma resumida la tabla está compuesta de la siguiente manera:

- Las filas están compuestas por los coeficientes y el término correspondiente a las restricciones de cada ecuación.
- La fila de costes, está formada por los coeficientes de la función objetivo.
- z_0 se trata del valor de la función objetivo evaluada en la solución básica admisible.

El siguiente paso es realiza iteraciones de tal forma que se vayan introduciendo en la base las variables que correspondan y sacando de la misma aquellas que no produzcan solución óptima. El algoritmo finalizará cuando los c' sean todos positivos en el caso de minimizar y negativos en el caso de maximizar.

Para más detalles en la aplicación del algoritmo consultar [11],[12].

3.3.2 Algoritmo de Johnson

Según [2] el algoritmo de Johnson es una forma de encontrar el camino más corto entre todos los pares de vértices de un grafo dirigido disperso. En este método se permiten a las aristas tener pesos o costes negativos aunque cabe destacar que no se permiten ciclos que consideren dichos costes o pesos. Se sirve del algoritmo de Bellman-Ford para hacer una transformación en el grafo inicial con el objeto de suprimir todas aquellas aristas que consten de peso negativo.

El algoritmo de Johnson se realiza en los siguientes pasos:

- En primer lugar se añade un nuevo nodo n al grafo , conectado a cada uno de los nodos del grafo por una arista cuyo coste es cero.
- El segundo paso consiste en aplicar el algoritmo de Bellman-Ford, comenzando por el nuevo nodo n. Esto tiene como objetivo conseguir determinar para cada vértice v el peso mínimo $h(v)$ del camino de n a v. En caso de detectar un ciclo negativo, el algoritmo finalizará.
- En tercer lugar, se les cambia el coste a las aristas del grafo original (aquel en el que no se encontraba el nodo n) por aquellos obtenidos mediante la aplicación del algoritmo de Bellman-Ford, obteniendo que: el nuevo tamaño de una arista comprendida entre u y v con tamaño $q(u,v)$ es $q(u,v) + h(u) - h(v)$.
- El último paso consiste en aplicar para cada nodo i el algoritmo de Dijkstra para determinar el camino más corto entre dicho nodo hacia los otros nodos, aplicando dicho algoritmo sobre el grafo cuyos pesos han sido modificados en los pasos anteriores.

Ejemplo:

Partimos del siguiente grafo:

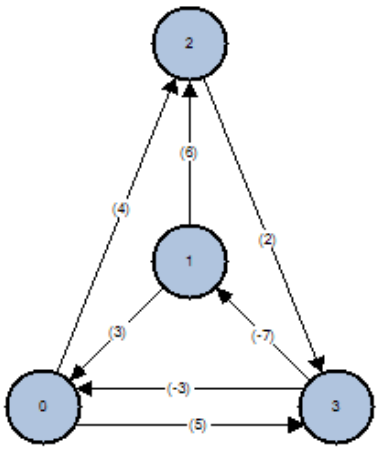

Ilustración 22: Aplicación del algoritmo de Johnson(I)

Paso inicial:

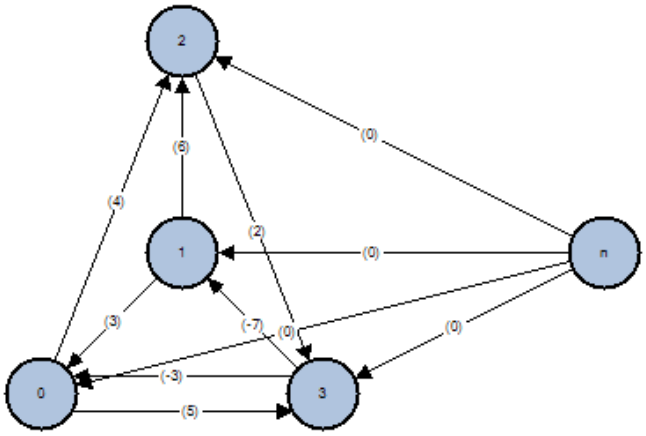

Ilustración 23: Aplicación del algoritmo de Johnson (II)

Pasos intermedios:

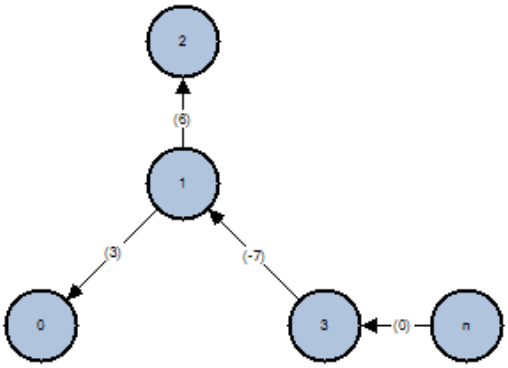

Ilustración 24: Aplicación del algoritmo de Johnson (III)

Finalización del algoritmo:

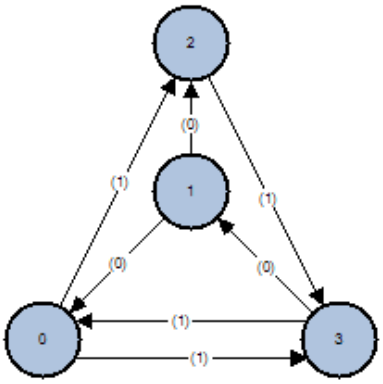

Ilustración 25: Aplicación del algoritmo de Johnson(IV)

Implementación: En [2] se encuentran aspectos útiles referentes a la implementación del algoritmo en varios lenguajes de programación así como el pseudocódigo que permite su diseño.

3.3.3 Algoritmo de Floyd-Warshall

El algoritmo de Floyd-Warshall se trata de un método que tras realizar análisis sobre grafos permite encontrar el camino mínimo en grafos dirigidos ponderados. En concreto, permite localizar el camino entre todos los pares de vértices en una única ejecución. Para ello, es necesario expresar el grafo mediante una matriz conocida como matriz de adyacencia quien representa si existe una unión o no entre nodos.

Consideraciones del algoritmo

Con este método se comparan todos los posibles caminos entre cada par de vértices, de tal forma que es capaz de hacerlo con sólo V^3comparaciones (puede haber hasta V^2 aristas en el grafo). El objetivo del algoritmo es obtener resultados mejores paulatinamente hasta localizar la solución óptima.

Sea un grafo G con conjunto de vértices V numerados hasta un total de N. Consideremos la existencia de una función en la forma **pathmin(i,j,k)** quien usando los vértices de 1 a k como puntos intermedios devolverá el camino mínimo de i a j. El objetivo será encontrar el camino mínimo usando los vértices de 1 hasta k+1.

Hay dos posibles soluciones: el camino que está formado por los vértices del conjunto desde 1 hasta k, o aquel que hay desde i hasta k+1 y de k+1 hasta j que es mejor solución. Teniendo en cuenta todo lo anterior, se sabe que el camino óptimo de i a j está definido por **pathmin(i,j,k)** y la longitud del camino sería la concatenación del camino mínimo de i a k+1 y el camino de k+1 a j.

Por tanto es posible definir de forma recursiva:

$$pathmin(i,j,k) = \min(pathmin(i,j,k-1), pathmin(i,k,k-1) + pathmin(k,j,k-1))$$

Esta expresión funciona ejecutando primero $pathmin(i,j,1)$ para todos los pares (i,j) continuando el proceso hasta $k = n$ localizando de esta forma el camino más corto para todos los pares de vértices.

Implementación: En [2] se pueden encontrar la implementación en varios lenguajes de programación.Para más detalles acudir a [13].

3.3.4 Algoritmo de búsqueda A^*

Con la aplicación del algoritmo A^*se encuentra (siempre y cuando se cumplan las condiciones), el camino de coste mínimo entre un nodo origen y un nodo destino u objetivo.

Consideraciones sobre el algoritmo

Este método emplea la heurística como herramienta para obtener su cometido. La heurística es una función matemática que podemos definir en la forma $h(n)$,con n siendo el número de nodos, que permite realizar una estimación del camino con coste mínimo de un nodo origen a un nodo destino. Para ello, se sirve de la búsqueda egoísta quien seleccionará el nodo cuyo valor sea el más bajo en lo que función heurística se refiere.

Tras ello, A^* expandirá los nodos que cumplen que $g(n) + h(n)$tienen el valor más bajo donde $g(n)$ contiene el valor extacto del coste entre el camino desde la situación de inicio hacia el nodo actual. En el momento en que $h(n)$ nunca sobreestime los costes para localizar el objetivo y sea ,por tanto, admisible A^* es óptimo.

El algoritmo consta de las siguientes propiedades:

- Como todo algoritmo de búsqueda A^*, se trata de un algoritmo completo ya que en caso de existir una solución, siempre será localizada.
- Si para todo nodo n del grafo $g(n) = 0$ nos encontramos ante una búsqueda voraz. En caso de cumplirse que $h(n) = 0$,entonces A^* pasa a ser una búsqueda de coste uniforme no informada.
- Para garantizar la eficiencia del algoritmo y su optimización, es necesario que $h(n)$ cumpla con ser una función heurística admisible, esto es, que no se hagan sobreestimaciones del coste real al alcanzar el nodo destino.

En caso de no cumplirse lo anterior, nos encontramos frente a un algoritmo denominado A quien, a pesar de ser completo, no asegura que se haya encontrado el camino con coste mínimo.

Debido al espacio necesario para su ejecución, hay variaciones de este algoritmo como son IDA^* o SMA^*.

Implementación: En [2] se pueden encontrar la implementación en varios lenguajes de programación.

3.3.5 Algoritmo Voraz

Según [2] un algoritmo voraz es aquel que para resolver un determinando problema sigue una heurística consistente en realizar la selección óptima en cada paso local con la esperanza de llegar a una solución general óptima. No presenta especiales dificultades en su diseño y nomalmente es aplicado a problemas de optimización.

Para su aplicación es necesario tener en cuenta siguiente:

Dado un conjunto finito de entradas E, un algoritmo voraz devolverá un conjunto S (conocidos como seleccionados) que pertenecen a E y que cumplen con todas las restricciones del problema. A cada conjunto de S que cumpla con las restricciones se les denomina conjunto prometedor y en el momento en que se logra minimizar o maximizar la función objetivo se dirá que S es el óptimo.

Funcionamiento del algoritmo

En cada paso se selecciona el mejor elemento $k \in E$ de entre los posibles denominado elemento prometedor. Se elimina del conjunto de candidatos y se comprueba si la inserción de dicho elemento procuce una solución admisible. Si no fuera factible se descarta el elemento y se itera de nuevo.En [14] se tienen más detalles de estos algoritmos.

3.3.6 Algoritmo de Bellman-Ford

El algoritmo de Bellman-Ford (BF) tiene como objeto obtener el camino más corto en un grafo dirigido ponderado. Su principal uso es debido a que funciona en grafos que contienen aristas con costes negativos.

Funcionamiento del algoritmo

Este algoritmo se basa en relajar todas las aristas $V - 1$ veces donde V es el número de vértices que contiene el grafo. Las iteraciones permiten a las distancias mínimas recorrer el árbol, ya que si no hay ciclos negativos, el camino más cortos solo visitará los vertices una vez. Existen dos versions del algoritmo:

- Versión no optimizada para grafos con ciclos negativos.
- Versión optimizada para grafos con aristas con coste negativo.

Implementación: En [2] se puede encontrar la implementación de dicho algoritmo en varios lenguajes de programación. Para más detalles puede acudir a [13].

Ejemplo:

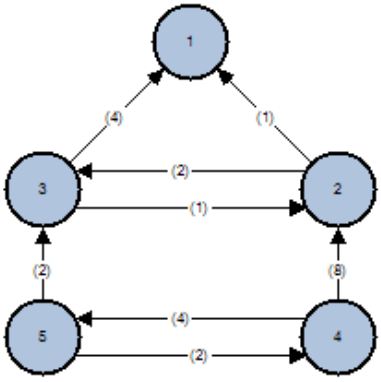

Ilustración 26: Previo a la aplicación del algoritmo BF

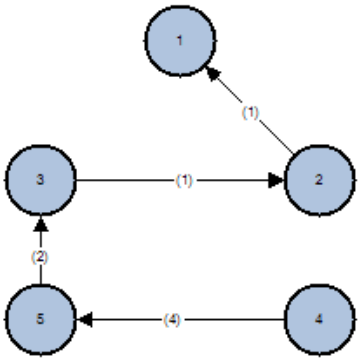

Ilustración 27: Resultado del algoritmo BF

3.3.7 Algoritmo de Dijkstra

Este algoritmo también conocido como algoritmo de ruta mínima permite conseguir el camino mínimo entre un origen y un destino en un grafo determinado. Su uso es principalmente en las redes de telecomunicaciones especialmente en el campo telemático reduciendo los tiempos de envío y recepción de datos entre un origen y un destino.

Funcionamiento del algoritmo

La idea fundamental del algoritmo es explorar los caminos más cortos desde un determinado origen hacia los demás nodos o vértices. Si es localizado dicho camino, el algoritmo finaliza con éxito. Destacar que el algoritmo no se puede aplicar a grafos con costes negativos. En términos generales:

Sea un grafo con N nodos conformando una red cerrada y sea q el nodo origen, D un vector de tamaño N que contendrá las distancias desde el nodo origen hacia el resto de nodos. Se aplican los siguientes pasos:

- Se comienza con todas las distancias en D con valor infinito ya que se desconocen en un principio, a excepción del nodo q cuya distancia hacia sí mismo es cero.
- A continuación se emplea un nodo a como nodo actual y se recorren todos los nodos adyacentes w_i siendo estos últimos los nodos no marcados.
- El siguiente paso consiste en calcular la distancia entre dicho nodo hacia sus nodos adyacentes mediante la expresión $distancia(w_i) = D_a + d(a, w_i)$ cuyo significado es que la distancia hacia el nodo no marcado w_i es la distancia del nodo actual D_a sumada a la distancia desde dicho nodo hacia el nodo w_i. Si $distancia(w_i)$ es menor a la distancia almacenada en el vector D entonces se actualiza dicho vector con la nueva distancia obtenida.
- Completado el paso anterior se marca el nodo a y se toma como siguiente nodo actual el de menor valor en el vector de distancias D regresando al segundo punto.

Ejemplo:

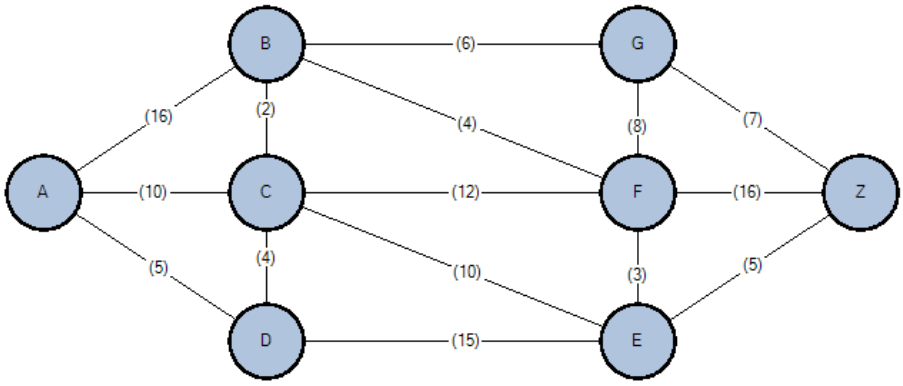

Ilustración 28: Aplicación del algoritmo de Dijkstra (I)

Se pretende obtener el camino mínimo entre el nodo origen A y el nodo destino Z. El resultado de aplicación del algoritmo de Dijkstra es el siguiente:

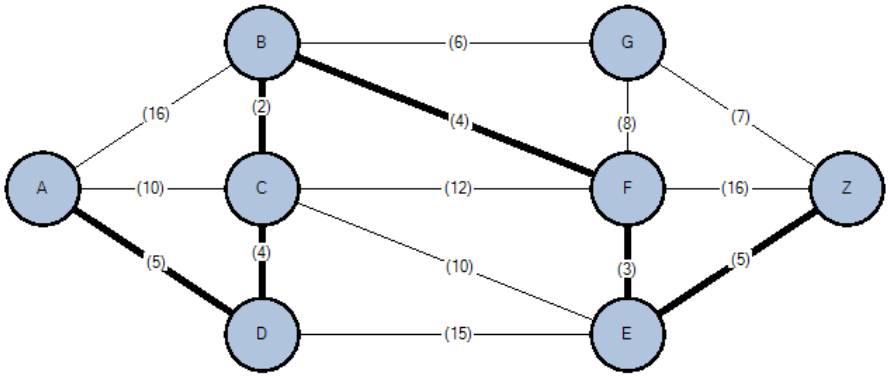

Ilustración 29: Aplicación del algoritmo de Dijkstra(II)

Implementación: En [2] se pueden encontrar la implementación de dicho algoritmo en varios lenguajes de programación. Para más detalles consultar [13].

3.3.8 Algoritmo Euleriano

Un grafo es Euleriano si tiene un circuito Euleriano. Se entiende por circuito Euleriano (en un grafo), aquel que recorre cada arista una y solo una vez. Destacar que si el grafo tiene al menos dos componentes conexas no triviales, no es posible que sea circuito Euleriano.

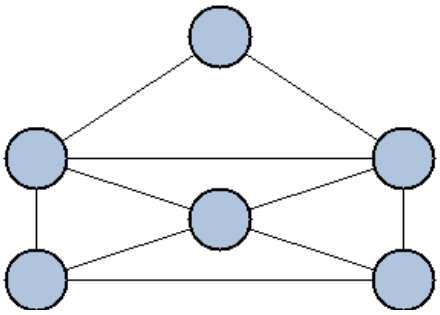

Ilustración 30: Grafo Euleriano

Descripción general

Dado $G(V, E)$ un grafo no orientado y conexo. Si tiene $2k$ nodos de grado impar, se dice que entonces G puede ser escrito como unión de k caminos distintos sobre los arcos, por lo que se puede decir que:

- G es euleriano.
- Todo vértice $v \in V$ tendrá grado $v \geq 2$.
- Todos los caminos del grafo serán distintos.

El algoritmo cuenta con las siguientes propiedades:

- Un grafo no dirigido es euleriano si es conexo y si se puede descomponer en uno con los vértices disjuntos.
- Un grafo conexo y no dirigido es euleriano si cada vértice tiene un grado par.
- Un grafo no dirigido se dice que es susceptible de ser recorrido, si es conexo y dos vértices en el grafo tienen grado impar.

Un ejemplo de aplicación típica del algoritmo euleriano es **"el problema del cartero chino".** Este problema se define de la siguiente forma:

Dado un grafo $G(V, E)$ con unos costes asignados a sus aristas (c_i), el problema del cartero chino consiste en encontrar un circuito de coste mínimo que pase por cada arista de G como mínimo una vez. Este algoritmo se trata con un algoritmo Euleriano si el grafo se trata de un circuito euleriano.

Implementación: En [2] se pueden encontrar implementaciones de varios métodos de resolución en lenguajes de programación.

4 ROBOT MÓVIL BASADO EN RASPBERRY PI

En los apartados anteriores se ha realizado una introducción sobre los aspectos teóricos necesarios que permiten entender y abordar el proyecto realizado. A partir de estas líneas, se van a detallar los pasos necesarios para la resolución del mismo. En concreto, se ha realizado el montaje de un vehículo robot basado en Raspberry Pi cuyo principal objetivo es que sea capaz de aplicar un algoritmo de planificación de rutas o path planning para llegar desde un origen específico a un destino determinado.

4.1. Montaje del Vehículo Robot

Para la realización del trabajo se ha dispuesto del siguiente material (Ilustración 31): "chasis de un vehículo robot con ruedas todoterreno accionadas con motores".

Ilustración 31: Piezas del vehículo robot

Sus características son las siguientes:

Tipo de Chasis	Chasis 4x4 compatible con placa Raspberry
Motores	4 motores micro-speed de alta calidad
Material de piezas	Integramente de aluminio
Posibilidad de incluirsensores	Sí
Alimentación de motores	3-12 V DC
Velocidad maxima	90 cm/s
Dimensiones	200x170x105 mm
Peso	660 gramos
Adicional	Incluye portapilas 4xAA y cables

Tabla 4: Características del vehículo robot

Para la realización del montaje, se han seguido las instrucciones proporcionadas por el fabricante y que son accesibles en [20]. La finalización del montaje da lugar a la Ilustración 32.

Ilustración 32: Montaje del vehículo robot

Consideraciones importantes: El vehículo robot solo puede mover los motores hacia delante o hacia atrás, por ello para realizar giros hacia un lado u otro se accionará un motor hacia delante y el otro hacia detrás y viceversa.

Para más detalles sobre las características o montaje del vehículo se puede acudir a [15].

4.2. Raspberry Pi

Para poder aplicar el algoritmo de planificación de rutas o path planning, es necesario disponer de un sistema microprocesador capaz de accionar los motores. Además dentro de las características de este sistema, se debía cumplir una fundamental: que el sistema fuera pequeño para que cupiese en el robot y que su peso no supusiese ningún problema para el movimiento del mismo. En definitiva, la placa Raspberry Pi cumple con todas las características necesarias para desarrollar todo lo referente al algoritmo de path planning.

La Raspberry Pi se trata de un computador de placa reducida de bajo costo, desarrollado por Reino Unido y cuyo principal objetivo es apoyar la enseñanza de las ciencias de la computación en las escuelas así como el aprendizaje del lenguaje de programación Python.

El software que emplea es "open source", siendo su sistema operativo oficial Raspbian (versión adaptada de Debian). Sus características principales son: incluye un procesador Broadcom, una memoria RAM, una GPU, puertos USB, HDMI, Ethernet, 40 pines GPIO y un conector para cámara. Como sistema de almacenamiento usa tarjetas SD, MicroSD.

4.2.1 Modelos de Raspberry Pi

Raspberry Pi 1 Modelo A, B y B+

Este modelo se caracteriza por no tener puerto ethernet siendo su conexión a internet a través de módulos WiFi del tipo USB. Tiene 26 conectores GPIO, salida de vídeo vía HDMI y vídeo RCA, conector Jack de 3.5 mm, un USB y conector de cámara. Tiene un procesador Broadcom VideoCore IV. Su alimentación es de 5 V y 2 A.

Los modelos B y B+ consisten en mejoras incluidas a la placa como son el aumento a 512MB de memoria RAM, USBs adicionales, la inclusión del puerto ethernet y el paso de memoria SD a microSD.

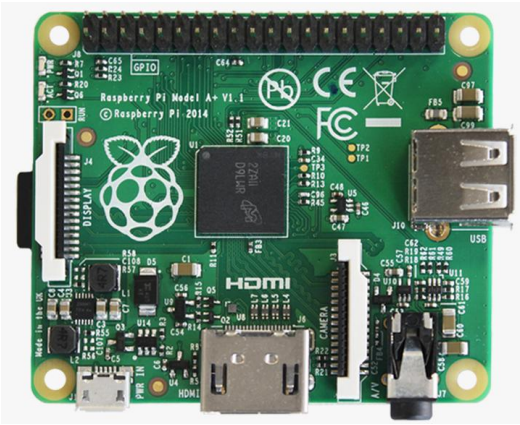

Ilustración 33: Raspberry Pi 1 Modelo A

Raspberry Pi 2 Modelo B

Este modelo viene marcado por el cambio del procesador. En concreto, se usa el modelo Broadcom BCM2836 de cuatro núcleos a 900 MHz. Este no fue el único cambio ya que se aumentó la memoria RAM a 1GB así como la aparición de hasta 40 pines GPIO. Se elimina la RCA y se mantienen del modelo anterior los 4 USBs.

Este modelo ha sido fundamental en el desarrollo de este proyecto, debido a que es el encargado de ejectuar el algoritmo de planificación de rutas o path planning, así como actuar sobre los motores para su desarrollo hardware.

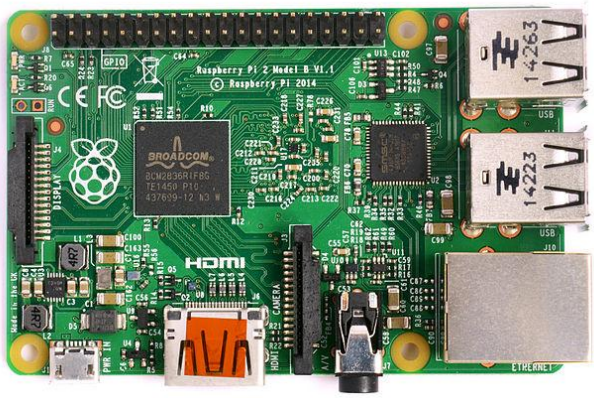

Ilustración 34: Raspberry Pi 2 Modelo B

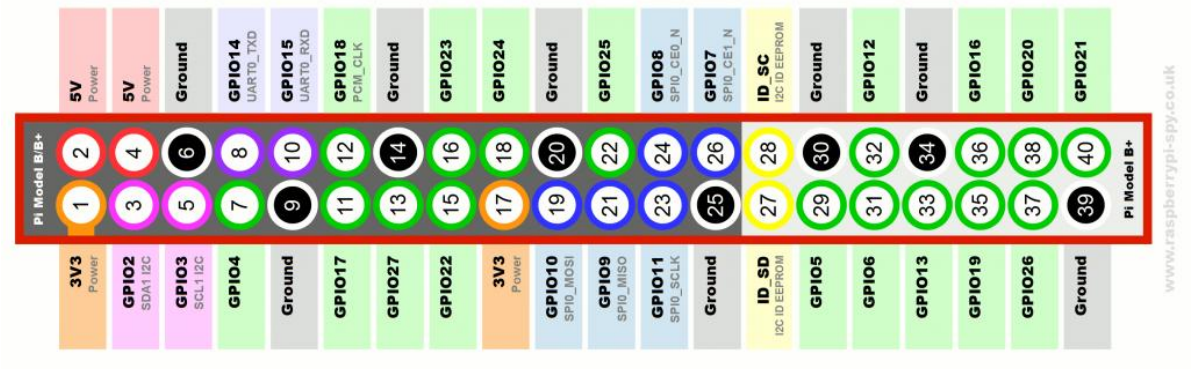

Ilustración 35: Pines GPIO Raspberry Pi 2 modelo b

Raspberry Pi 3 Modelo B

Se trata de la versión más reciente hasta el desarrollo de este proyecto. Caracterizada por usar un nuevo procesador, en concreto Broadcom de cuatro núcleos a 1.20GHz. Mantiene su RAM e incorpora la inclusion de módulos WiFi y Bluetooth.

Su aplicación ha sido fundamental en el desarrollo de este trabajo, debido a que se ha utilizado para realizar medidas de posicionamiento global del robot mediante un módulo GPS.

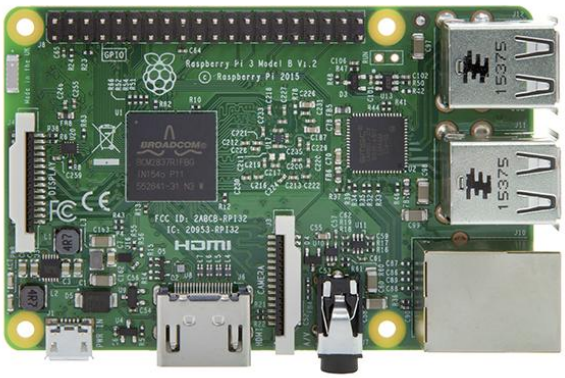

Ilustración 36: Raspberry Pi 3 Modelo B

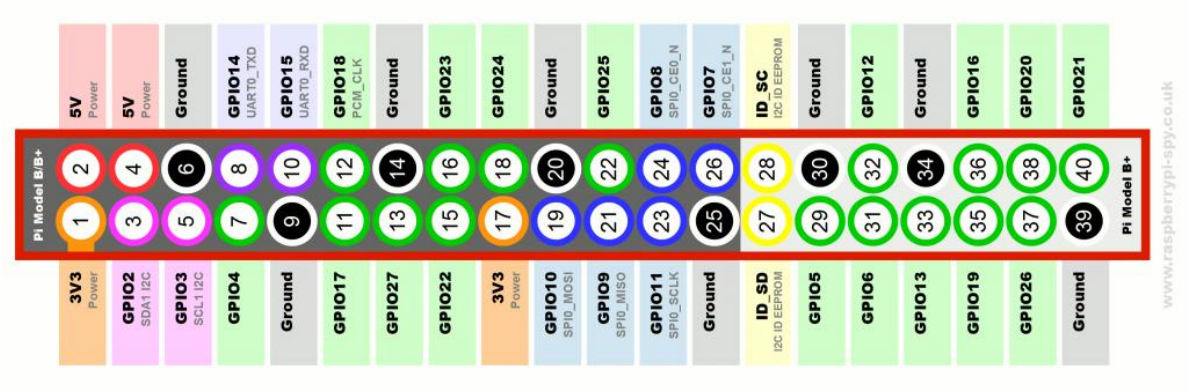

Ilustración 37: Pines GPIO Raspberry Pi 3 Modelo B

4.2.2 Instalación y configuración del sistema operativo

Como se indicó, el sistema operativo oficial de la Raspberry Pi es Raspbian. Para su instalación, se acude a [22] en concreto al apartado de descargas para obtener "NOOBS". A continuación se realizan los siguientes pasos:

1. Insertar SD en el PC donde se haya descargado "NOOBS".
2. Formatear la tarjeta SD.
3. Descomprimir "NOOBS" en dicha tarjeta SD.
4. Insertamos la tarjeta SD en la Raspberry Pi y se siguen las instrucciones de instalación.

Tras finalizar la instalación, aparecerá la interfaz gráfica de Raspbian como se muestra en la Ilustración 38.

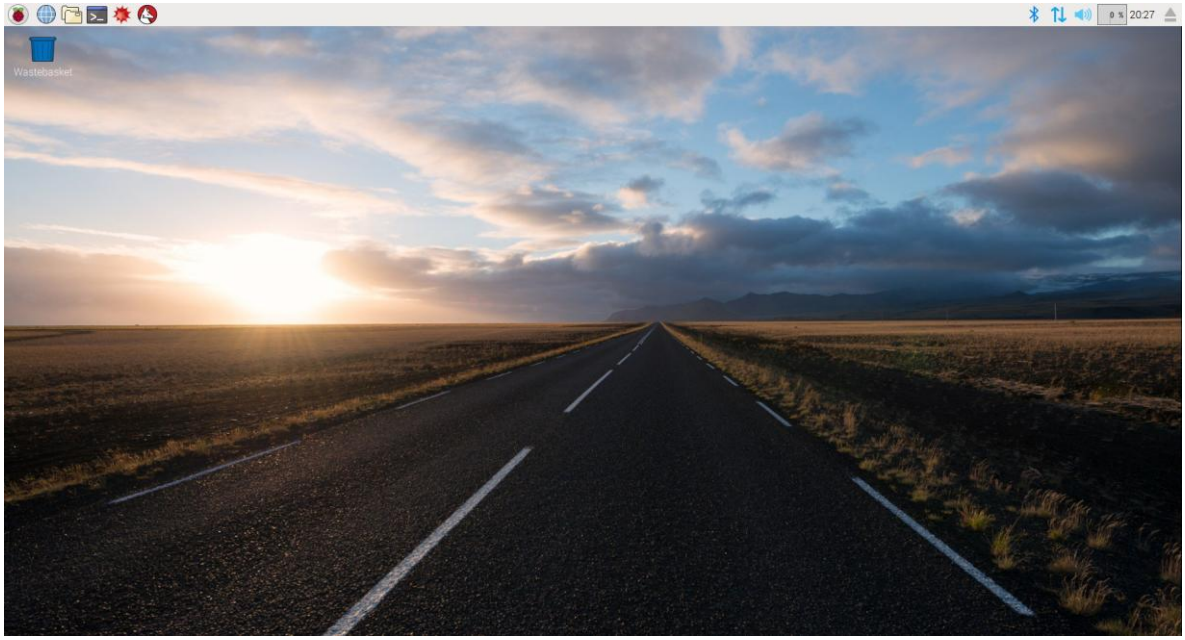

Ilustración 38: Interfaz gráfica de Raspbian

Tras finalizar la instalación del sistema operativo es fundamental comprobar que esté actualizado. Para ello, se abre un terminal y se ejecutan los siguientes comandos:

```
sudo apt-get update
sudo apt-get upgrade
```

4.2.3 Acceso remoto a Raspbian

Para tratar el vehículo móvil será necesario un acceso remoto mediante WiFi. Para ello, se dispone de un módulo que actúa como receptor cuyo nombre es WiPi. El único elemento necesario es un router que permita la comunicación remota y para ello se utilizará en este proyecto un smartphone con capacidad de habilitar su red móvil como router. A continuación, se conectará un PC al smartphone para estar en la misma subred que la raspberry pi y se accederá a esta mediante el cliente VNC (debe estar habilitado en la raspberry pi).

4.2.4 Alimentación

Para que el vehículo robot pueda moverse sin limitaciones, se usarán dos baterías portátiles (una que para cada Raspberry) con una tension de salida de 5V y una autonomía de aproximadamente una hora.

4.3. Drivers de los motores

Para la aplicación del algoritmo de planificación de rutas o path planning en la realidad, es necesario un elemento que permita accionar los motores a través del sistema microprocesador empleado (en este caso a través de la Raspberry Pi). Para ello, se ha utilizado una placa que consiste en un controlador de motores (en concreto el modelo DRV8835) cuyo funcionamiento consiste en traducir las órdenes recibidas (mediante sus librerias) a tensiones y corrientes aplicadas a determinados GPIO que se encargarán de accionar los motores. La placa se encuentra disponible en **¡Error! No se encuentra el origen de la referencia.** y su principio de funcionamiento básico es la modulación por anchura de pulsos (PWM).

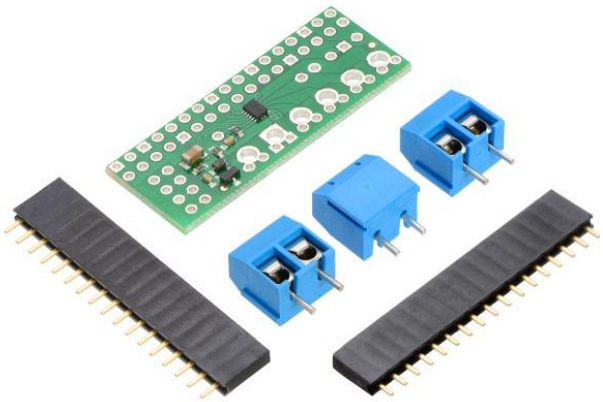

Ilustración 39: Controlador DRV8835

En la siguiente tabla se encuentran resumidas las características principales del controlador DRV8835:

Tipo de Puente	Doble Puente H capacidad de controlar dos motores DC o un motor paso a paso bipolar.
Alimentación del motor	1.5V a 11V.
Corriente de salida	1.2A continuo(1.5 A de pico) por motor.
Alimentación de lógica	2V a 7V.
PWM	Soporta hasta 250KHz.
Modos de funcionamiento	PHASE/ENABLE (por defecto, un pin de dirección y otro de velocidad) o IN/IN (salidas en modo espejo).
Programación	Librerías Python.
Datosadicionales	Protección de polaridad en la entrada de alimentación.Protección de voltaje y temperatura.

Tabla 5: Características principales del controlador DRV8835

Para más detalles de las características técnicas puede visitar el datasheet[21].

Funcionamiento del controlador DRV8835

La placa utiliza las clavijas GPIO 5, 6, 12 y 13 para controlar el controlador del motor, haciendo uso de las salidas PWM del hardware de Raspberry Pi, aunque las asignaciones de pines pueden personalizarse si los valores por defecto no son adecuados. En términos generales el funcionamiento es el siguiente:

La placa es alimentada a través de uno de los pines de la Raspberry Pi que proporciona 3.3V. Para que esto sea posible, la Raspberry debe ser alimentada previamente a través de su terminal de alimentación MicroUsb y la placa del motor debe estar suministrada con una tensión que comprende desde 1.5V a 11V a través de sus terminales VIN y GND. El controlador proporciona tres orificios pasantes por donde se puede conectar si fuera necesario un regulador de tensión. En la Ilustración 40 se encuentran en detalles los pines de la placa.

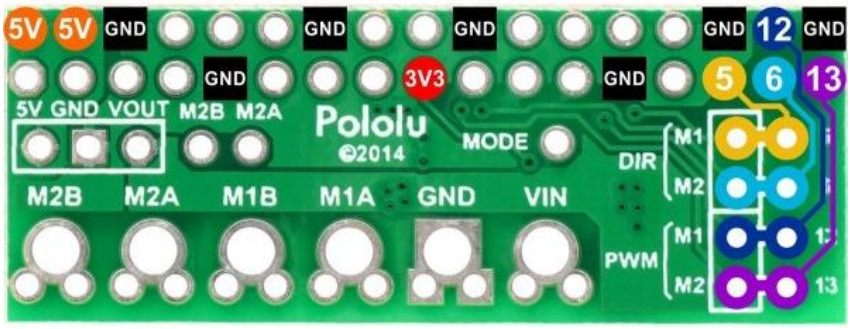

Ilustración 40: Situación de pines del controlador DRV8835

Por defecto, el controlador viene configurado para operar en modo PHASE/ENABLE donde una señal PWM aplicada al pin ENABLE determinará la velocidad del motor, y será el estado digital del pin PHASE quien determinará el sentido de rotación del mismo. Los GPIO 12 y 5 se usarán para controlar la velocidad y la dirección respectivamente del motor 1 y los GPIO 13 y 6 para controlar la velocidad y dirección del motor 2. En la Tabla 6 se muestra como afectan las salidas a este modo:

Xfase	**Xenable**	**MxA**	**MxB**	**Modo**
1	PWM	L	PWM	Marcha atrás
0	PWM	PWM	L	Adelante
X	0	L	L	Freno

Tabla 6: Funcionamiento modo Phase/Enable

Este modo de funcionamiento es el adecuado para la mayoría de aplicaciones. En concreto, éste ha sido el modo de funcionamiento empleado en este proyecto.

Otro modo de funcionamiento disponible es el modo IN/IN que permite opciones de control algo más avanzadas según recoge la siguiente tabla:

XIN1	**XIN2**	**MxA**	**MxB**	**Modo**
0	0	ABIERTO	ABIERTO	Salidas apagadas
0	PWM	L	PWM	Reverso
PWM	0	PWM	L	Adelante
PWM	1	L	NOT PWM	Marcha atrás
1	PWM	NOT PWM	L	Adelante
1	1	L	L	Freno

Tabla 7: Modo de funcionamiento IN/IN

Para usar este modo, es necesario configurar mediante software el PWM y es utilizado cuando interesa controlar encendido/ apagado de los motores.

Soldadura

Para poder conectar la placa a la Raspberry es necesario realizar la soldadura de la misma a los conectores que permitirán su adaptación a los GPIO de la raspberry pi. Por otra parte, es necesario realizar la soldadura de los pines que permitirán realizar el cableado de los motores, así como de la alimentación (mediante las pilas). El resultado puede verse en la Ilustración 41.

Ilustración 41: Detalles de la soldadura del controlador DRV8835

Tras realizar las soldaduras, es posible realizar el acoplo del controlador DRV8835 a la raspberry pi sin más que conectarlo a los GPIO en la forma que se muestra en la Ilustración 42.

Ilustración 42: Raspberry Pi con controlador DRV8835

Pruebas de funcionamiento de los motores

Como se comentó en el comienzo de este apartado, la forma que tiene el usuario de programar el comportamiento de los motores es mediante el uso de unas librerías en código Python proporcionadas por el fabricante. Estas librerías, permiten la programación de una forma relativamente sencilla y se pueden localizar en [23], siendo los pasos necesarios para su instalación los siguientes comandos en un terminal:

1. Se instala WiringPi-Python:

```
sudo apt-get install python-dev python-pip
sudo pip install wiringpi
```

2. Se descarga e instala la librería pololu_drv8835_rpi:

```
git clone https://github.com/pololu/drv8835-motor-driver-rpi.git
cd drv8835-motor-driver-rpi
sudo python setup.py install
```

De la librería podemos destacar los siguientes métodos disponibles:

- `motors.setSpeeds(m1_speed,m2_speed)` que permite establecer la velocidad y dirección de ambos motores.
- `motors.motors1.setSpeed(speed)` que permite establecer la velocidad y dirección del motor 1.
- `motors.motors2.setSpeed(speed)` que permite establecer la velocidad y dirección del motor 2.
- La aparición de una constante "`MAX_SPEED`" que es muy útil para establecer la velocidad.

Todo lo anterior es referente a la parte software pero para poder probar los motores, éstos deben estar conectados al controlador. Para ello, es necesario seguir el siguiente esquema:

Ilustración 43: Conexionado de motores al controlador drv8835

Como se puede apreciar en la Ilustración 43, por cada controlador se pueden accionar hasta dos motores razón por la que se ha decidido usar sólo dos ruedas motrices.Tras realizar dicho conexionado así como todos los pasos previos (como la instalación de librerías), se puede ejecutar un ejemplo dado por el fabricante que consiste en mover hacia delante durante 5 segundos y hacia atrás durante otros 5 segundos ambos motores haciendo lo siguiente:

```
sudo python example.py
```

4.4. GPS

El Sistema de Posicionamiento Global (GPS) es un sistema que permite determinar en toda la Tierra la posición de un objeto con una precisión de hasta centímetros[2].En nuestro caso, se usará un módulo GPS (en concreto el modelo Microstack GPS L80) compatible con Raspberry Pi.

Ilustración 44: Módulo Microstack GPS L80 para Raspberry Pi

Las características principales del módulo GPS L80 son las siguientes:

- Baja potencia (25mA en modo de adquisición, 20 mA en modo de seguimiento y hasta 3mA en modo "AlwaysLocate").
- Fijación en tiempo corto.
- Receptor de 1575.42MHz con 22 canales para seguimiento así como 66 canales para adquisición de datos.
- Compatible con DGPS, SBAS (WASS, EGNOS/MSAS/GAGAN).

- Registro de datos integrados.
- Permite programación de modos (activación/suspensión del módulo GPS).
- Alimentación 3.3V.

Previo al conexionado con la Raspberry Pi (en este proyecto se ha usado el modelo 3 B) es necesario tener en cuenta las siguientes consideraciones:

La Raspberry Pi 3 tienen dos interfaces series:

- /dev/ttyAMA0 utilizado por el Bluetooth.
- /dev/ttyS0 asociado al GPIO.

Por defecto, el puerto serie está deshabilitado. Para habilitarlo se realiza lo siguiente a partir de una consola de comandos:

```
sudo nano /boot/config.txt
```

y se agrega en la parte inferior:

```
enable_uart=1
```

para que se guarden los cambios se reinicia el sistema mediante:

```
sudo reboot
```

Referente al uso del módulo GPS , es necesario saber que el puerto serie /dev/ttyS0 por defecto está asignado a los pines GPIO 14 y 15 que son necesarios para la comunicación entre dicho módulo y la Raspberry Pi. Para que el módulo GPS pueda acceder sin problemas al puerto serie, es necesario desactivar la consola ttyS0 mediante los siguientes comandos:

```
sudo systemctl stop serial-getty@ttyS0.service
sudo systemctl disable serial-getty@ttyS0.service
```

Tras su realización será necesario acceder al fichero "cmdline.txt" mediante:

```
sudo nano /boot/cmdline.txt
```

Para eliminar lo siguiente "console = serial0,115200". Tras su realización se guarda el documento y se reinicia el sistema para guardar los cambios realizados.

A continuación, ya es posible instalar el software que necesitará disponer el módulo GPS para funcionar. Se deben realizar los siguientes pasos:

1. `sudo apt-get update`
2. `sudo apt-get upgrade`
3. `sudo raspi-config`

   Pulsar intro en "Advanced Options" para luego hacerlo sobre "Serial" y aparecerá el siguiente mensaje "Would you like a login Shell to be accesible over serial?". Seleccionar "Yes".
4. `sudo apt-get install python3-microstacknode`
5. `sudo apt-get install gpsdgpsd-clients python-gps`

6. Accedemos a al fichero gpsd mediante `sudo nano /etc/default/gpsd` y se establece la siguiente configuración:
   - `START_DAEMON= "true"`
   - `USBAUTO= "false"`
   - `DEVICES= "/dev/serial0"`
   - `GPSD_OPTIONS= "-n -G"`
7. Por último, se proporcionan los siguientes permisos `sudo chmod 775 ../../dev/ttyS0`
8. Realizar el montaje hardware.

En la siguiente Ilustración (dada por el fabricante en [24]) se puede apreciar los pines donde se debe conectar el módulo GPS para su funcionamento.

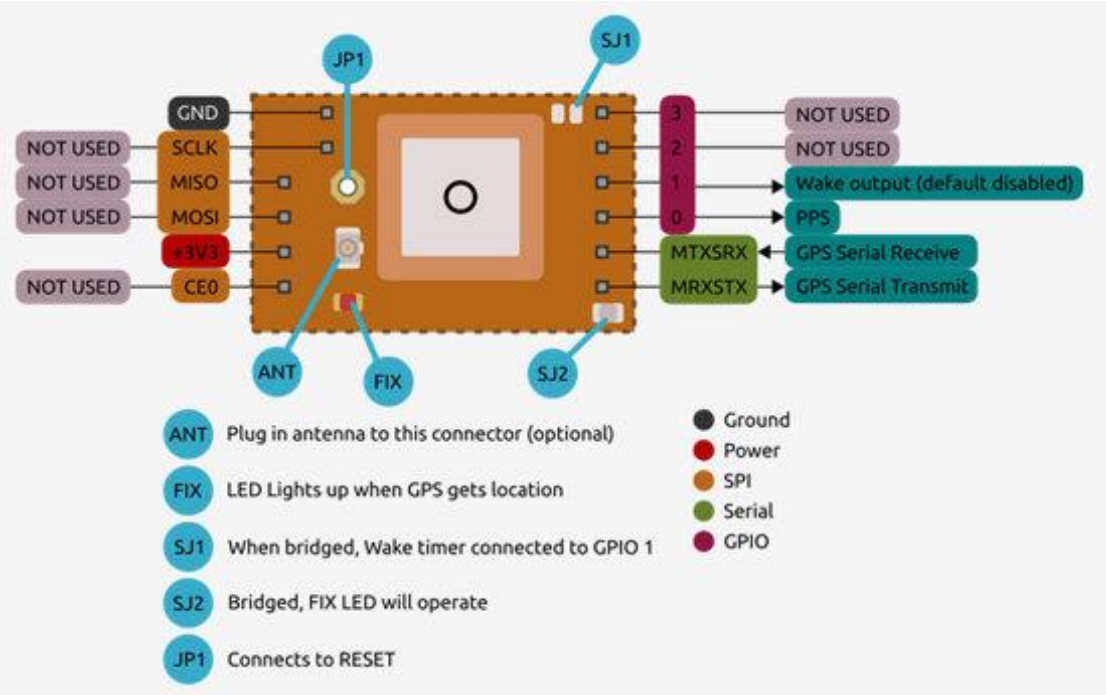

Ilustración 45: Conexionado del módulo GPS L80

De forma resumida, atendiendo a los GPIO de la Raspberry Pi que se utilice (en este proyecto se ha usado la Raspberry Pi 3 Modelo B para el módulo GPS) se debe conectar el GPIO MTXSRX (transmisor del puerto serie) al receptor del GPS y el MRXSTX (receptor del puerto serie) al transmisor del GPS. Tras pasar unos minutos deberá parpadear el led denominado en el esquema superior "FIX" indicando que ha detectado señal de GPS. Se puede ejecutar el comando `cgps -s` para verificar su funcionamiento.

Durante la realización del proyecto, se trató de probar el módulo GPS L80 mediante conectores de prueba pero no se obtuvierón medidas (no era capaz de obtener cobertura y por tanto el led FIX no parpadeaba) por parte del sistema.

Tras buscar información sobre el módulo se destacaba que éste era sensible a ESD (descargas electrostáticas) por lo que se recomendaba tener especial cuidado con los contactos así como el uso de una placa de expansión para Raspberry Pi donde se facilitaba todo el conexionado y los problemas de ESD eran reducidos.

Por tanto, se tomó la decisión de adquirir dicha placa de adaptación para no dañar el módulo. En la Ilustración 46 se muestra el esquema de la placa de adaptación.

Mediante el uso de dicha placa, tan solo era necesario conectar el módulo GPS L80 en su zona correspondiente (en la parte del SPI) y conectar la placa de adaptación a la Raspberry Pi 3 a través de sus GPIO como se puede ver en la Ilustración 47.

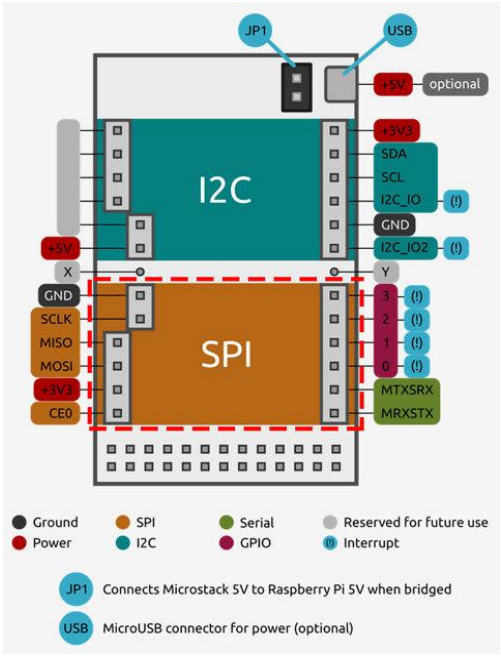

Ilustración 46: Esquema de microstack baseboard

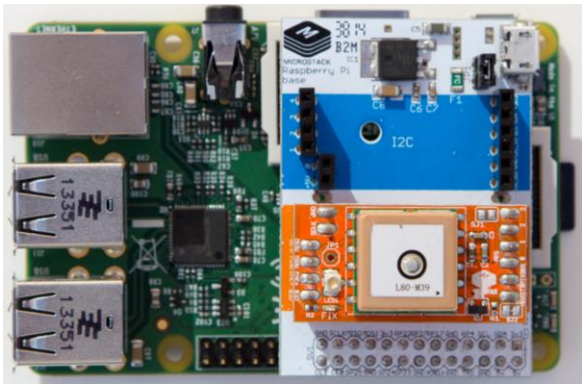

Ilustración 47: Conexión módulo L80 mediante la placa de adaptación

Tras realizar todos los pasos anteriores y obtener la luz "FIX" del módulo GPS parpadeando, se puede verificar su funcionamiento mediante el comando `cgps -s` tal como se muestra en la Ilustración 48.

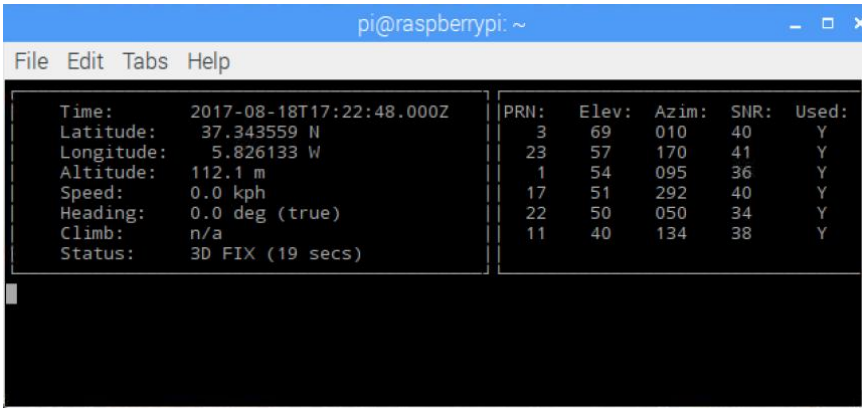

Ilustración 48: Prueba de funcionamiento del módulo GPS L80

4.5. Esquema general del sistema completo

En la Ilustración 49, tenemos el esquema general del sistema completo. Como se puede apreciar, el sistema está compuesto por el chásis del vehículo robot, una Raspberry Pi 2 modelo b que se encargará de controlar los motores y una Raspberry Pi 3 modelo b que se encargará de obtener los datos del módulo GPS L80.

La primera Raspberry estará ubicada en la parte central del vehículo robot mientras que la segunda estará en la plataforma superior para conseguir de una forma más sencilla la cobertura del satélite GPS. Para establecer control remoto sobre el vehículo robot, se utiliza un smartphone con capacidad de actuar como router y mediante un cliente VNC se puede acceder en remoto a ambas Raspberry para realizar las consultas oportunas o aplicar el algoritmo de path planning.

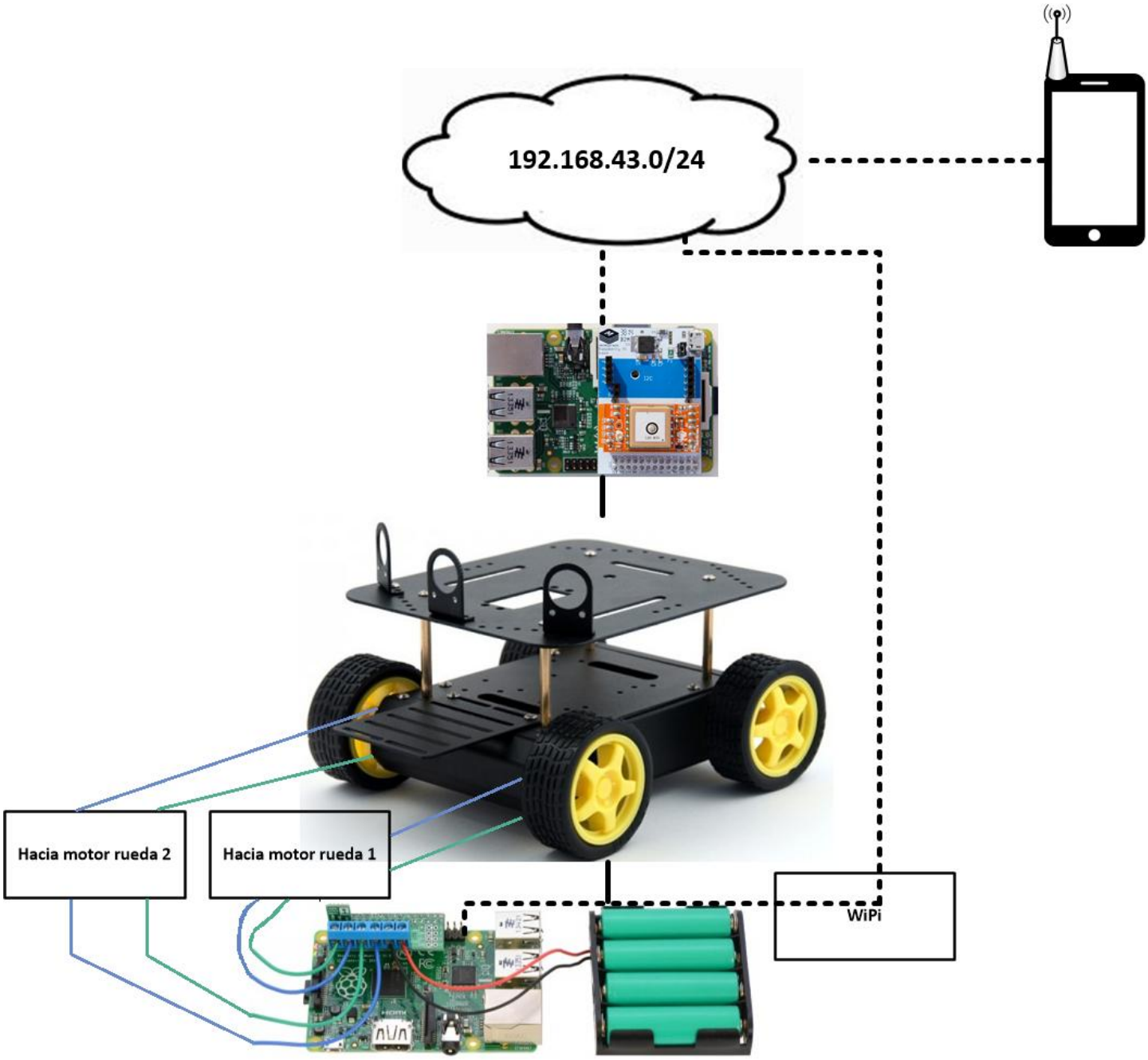

Ilustración 49: Esquema general del sistema completo

5 APLICACIÓN DE PATH-PLANNING

Una vez concluido el montaje del vehículo robot y la configuración de sus elementos (Raspberry Pi, controlador DRV8835 y módulo GPS L80) se procede al diseño y posterior programación del algoritmo de planificación de rutas o path planning mediante el lenguaje python. Con el objeto de facilitar las labores de programación se ha empleado un entorno cuyas herramientas son muy útiles fundamentalmente cuando se va a depurar el código. Este entorno se denomina "Anaconda" y se trata de una herramienta "open source" disponible para múltiples plataformas ofreciendo al usuario facilidades para programar en python. En [25] hay disponible más información sobre dicho entorno.

5.1. Planteamiento del problema

El problema a resolver es el siguiente: **Dado un terreno en ausencia de obstáculos y completamente llano, localizar un camino (siempre que exista) entre un origen y un destino seleccionados por el usuario**.

Para abordar dicho problema se dispone de un vehículo robot como el descrito en la Ilustración 49. Como se ha descrito previamente, se sabe que el vehículo robot solo puede realizar movimientos hacia delante y hacia atrás. Este aspecto es especialmente relevante para el tratamiento de los giros del vehículo ya que para acometerlos se deberá establecer el motor de una rueda hacia delante y el otro hacia detrás para obtener el giro hacia la izquierda y viceversa para obtener el giro hacia la derecha.

Teniendo en cuenta lo anterior, la decisión que se ha tomado para resolver el problema es la siguiente:

Para abordar el lugar:

- El terreno se puede modelar como un grafo. Este puede ser de cualquier tipo (véase el apartado **3.2.2 Representación mediante grafos**).
- Cada nodo del grafo constuirá un posible origen o destino.
- Los costes entre los nodos del grafo constituyen la distancia (en centímetros) que hay entre ambos.

Con respecto al vehículo robot:

- Por cada movimiento que tenga que realizar se representará con un nodo.
- En cada nodo, el vehículo podrá realizar los siguientes movimientos:
    - Movimiento hacia delante.
    - Movimiento hacia atrás.
    - Giro hacia la derecha.
    - Doble giro hacia la derecha.
    - Triple giro hacia la derecha.
    - Giro hacia la izquierda.
    - Doble giro hacia la izquierda.
    - Triple giro hacia la izquierda.
    - Parar el vehículo.
- Antes de realizar la actuación sobre los motores del robot, el algoritmo calculará el tiempo que deben estar funcionando.

Con todo esto establecido, sólo falta decidir el algoritmo de path planning que se va a emplear. De entre todos los tratados previamente podemos destacar el algoritmo de Dijkstra quién proporciona la ruta mínima en un grafo cerrado. Pero este algoritmo no se puede aplicar a cualquier tipo de grafo.

Por otra parte, tras realizar estudios a posteriori del movimiento del vehículo robot se pudo apreciar como éste no realizaba los movimientos como se esperaban (especialmente el giro a la izquierda). Estos motivos llevaron a plantear un algoritmo que se sirve de aspectos teóricos de algunos de los nombrados pero que toma decisiones nuevas como las siguientes:

- Si estando en un determinado nodo el vehículo robot puede ir al destino, se irá sin tener en cuenta el coste.
- Si estando en un determinado nodo el vehículo robot aún no puede alcanzar al destino, entonces irá hacia aquel nodo que presente coste mínimo.

De esta forma, a pesar de que el vehículo no siempre va a usar la ruta mínima, se garantiza que el margen de error sea el menor posible ya que en su mayor parte el vehículo se moverá hacia delante y precisamente es en este movimiento donde presenta el menor error de desplazamiento.

Por decisión propia, se ha dado a dicha resolución la denominación de **algoritmo Yowi**.

5.2. Resolución mediante el algoritmo Yowi

El algoritmo Yowi se sirve de un conjunto de métodos programados para obtener (siempre que exista) un camino entre un origen y un destino proporcionados por el usuario. En los sucesivos apartados se irán detallando el comportamiento de cada una de las funciones del algoritmo.

5.2.1. Base de datos del sistema

El algoritmo se sirve de una base de datos proporcionada por el usuario y que consiste en un fichero de texto plano con los siguientes campos:

Origen	Destino	C0	C1	C2	Dirección desde Nodo A	.....	Dirección desde Nodo Z

Tabla 8: Campos de la base de datos del sistema

Por cada línea, se pondrá la conexión de un determinado origen hacia un determinado destino, la distancia que los separa en centímetros (conformando las posiciones C0-C2) y por último se completaría la columna Dirección desde Nodo X diciendo los movimientos que debe realizar el robot para ir desde el origen hasta el destino si parte desde dicho nodo.En concreto se programará:

- 0 para indicar que el vehículo se pare.
- 1 para indicar que el vehículo se mueva hacia delante.
- 2 para indicar que el vehículo realice un giro hacia la derecha y continúe avanzando.
- 3 para indicar que el vehículo realice un giro hacia la izquierda y continúe avanzando.
- 4 para indicar que el vehículo se mueva hacia detrás.
- 5 para indicar que el vehículo realice un doble giro hacia la derecha y posteriormente continúe avanzando.
- 6 para indicar que el vehículo realice un triple giro hacia la derecha y posteriormente continúe avanzando.
- 7 para indicar que el vehículo realice un doble giro hacia la izquierda y posteriormente continúe avanzando.
- 8 para indicar que el vehículo realice un triple giro hacia la izquierda y posteriormente continúe avanzando.

Por ejemplo:

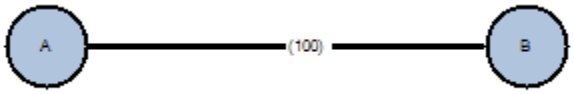

Ilustración 50: Ejemplo de base de datos

Origen	Destino	C0	C1	C2	DirecciónDesdeNodo A	DirecciónDesdeNodo B
A	B	1	0	0	1	X
B	A	1	0	0	X	5

Tabla 9: Ejemplo de base de datos

De la base de datos anterior se obtiene que para ir desde A hacia B el coste es 100 y la decisión que debe tomar el robot es ir hacia delante. En la segunda fila se indica que para ir desde B hacia A el coste es 100 y la decisión para tomar es realizar doble giro hacia la derecha e ir hacia delante.

Para evitar confusiones en las columnas donde establecemos la decisión que debe tomar el robot, se establece que antes de que se vaya a iniciar el algoritmo, el robot debe estar mirando hacia el este del nodo. Traducido a la Ilustración 50, si el robot se coloca en el nodo A, éste debe estar mirando hacia el este del nodo, hacia B, y si por el contrario el robot se coloca en B, éste debe estar mirando hacia su este. Por éste motivo se indica que si se parte de B y se va hacia A se debe realizar un doble giro en lugar de ir hacia delante.

Nota importante: Los nodos terminales , es decir, aquellos que no tienen un camino por donde salir deben tener una línea de la siguiente forma (suponemos que nodo F es nodo terminal):

Origen	Destino	C0	C1	C2	DirecciónDesdeNodo A	DirecciónDesdeNodo B
F	F	0	0	0	0	0

Con esto simplemente se indica que efectivamente el nodo F existe en el grafo pero es terminal.

5.2.2 Métodos de validación de la base de datos

Estos métodos (**validar** y **validar_direccion**) sirven para corroborar que la base de datos que proporciona el usuario es válida desde el punto de vista del contenido de la misma.

La función **validar** recibe el fichero que debe validar y realiza lo siguiente (para cada línea del fichero) :

1. Comprueba que el origen se encuentra en la primera posición de la línea y que además es una letra comprendida entre A-Z.
2. Comprueba si el destino se encuentra en la tercera posición de la línea y que además es una letra comprendida entre A-Z.
3. Comprueba que el coste se encuentra en la quinta, sexta y séptima posición y además cada posición contiene un carácter numérico.

El método **validar_direccion** hará lo propio pero con las posiciones que recogen las direcciones que debe tomar el robot comprobando que se correspondan con un carácter numérico.

En caso de encontrar alguna anomalía en la base de datos informará y se finalizará el programa.

5.2.3 Método de ordenación de la base de datos

Este método cuyo nombre es **ordenar** recibe el nombre del fichero que debe ordenar y lo reestructura obteniendo un fichero con denominación "fichero_ordenado.txt" y que recoge las líneas del fichero ordenadas alfabéticamente. Esto se hace por convención, el algoritmo funcionará de igual forma.

5.2.4 Métodos de lectura de la base de datos

Estos métodos (**lectura** y **lectura_direccion**) pasan al programa cada campo que contienen las líneas de la base de datos para su posterior procesado.En concreto:

El método **lectura** recibe como parámetro el "fichero_ordenado.txt" y realiza lo siguiente (por cada línea):

1. Almacenar campo origen.
2. Almacenar campo destino.
3. Almacenar campo coste.
4. Una vez finalizadas las líneas retornar dichos valores.

El método **lectura_direccion** por su parte recibe el "fichero_ordenado.txt" y el origen que haya establecido el usuario con el siguiente objetivo: retornar la columna completa correspondiente al conjunto de direcciones que debe tomar el robot partiendo de ese origen.

5.2.5 Método de simulación

Este método es fundamental ya que va a ser el que decida si el algoritmo se aplica en la realidad o no. Se trata de un método que realiza una simulación para saber si es posible localizar un camino (sirviéndose de los campos proporcionados por la base de datos) que permita llegar desde un origen hasta un destino dados por el usuario.

En caso afirmativo, el algoritmo devolverá las pautas que debe emplear el robot para acometerlo en la realidad. En caso negativo, se informará que no es posible localizar un camino que lleve desde el origen hacia el destino solicitado y se dejará al usuario decidir si controlar de forma remota el comportamiento del robot o finalizar el algoritmo.

El método de simulación tiene la siguiente forma:

*(destino_robot,direccion_robot,costes_robot)=**simular**(origen,destino,costes,direccion,origenuser,destinouser)*

recibe los campos proporcionados por la base de datos (origen,destino,costes,direccion) así como el origen y destino seleccionados por el usuario (origenuser y destinouser). Retorna los datos que posteriormente determinarán el comportamiento del robot, es decir, los destinos que debe ir visitando el robot (destino_robot), las direcciones que debe tomar el robot (direccion_robot) y los costes de cada camino que tomará el robot (costes_robot). Si el método no encuentra ninguna solución se retornarán estos tres parámetros con valor cero.

La esencia de esta función reside en la aplicación de otros dos métodos: **calcula_ruta** y **recalcula_ruta** cuyas formas se presentan de la siguiente manera:

*solucion=**calcula_ruta**(origen,destino,costes,direccion,origenuser,destinouser,origen_anterior)*

*ruta_recalculada=**recalcula_ruta**(caminos,origen_anterior,destino_robot,direccion_robot,costes_robot)*

El primero de ellos recibe como parámetros los datos proporcionados por la base de datos (origen,destino,costes y direccion), el origen (origenuser) establecido por el usuario y el destino (destinouser). Por último, recibe los lugares visitados, y que no se volverán a visitar (origen_anterior) con el objeto de evitar posibles bucles. Devolverá una lista con la posible solución encontrada o con información por si hubiera que recalcular ruta.

El segundo método sólo actúa siempre que no se encuentre solución en **calcula_ruta** y existan caminos alternativos. Esta función recibe el número de caminos que presenta cada nodo visitado (caminos), los lugares visitados y que no se volverán a visitar (origen_anterior) y las decisiones que en un principio se habían tomado para el robot (destino_robot, dirección_robot, costes_robot).

Con estos datos se trata de buscar cual de los nodos visitados (comenzando desde el último) tiene más de un camino (para regresar al mismo) y buscar por su camino alternativo. Por otra parte, si encuentra dicho nodo este método rectifica los valores referentes a las decisiones tomadas por el robot eliminando aquellas que no sirvan. Retorna el nodo desde el que se volverá a simular y los parámetros del robot rectificados.

En las siguientes ilustraciones se presentan los diagramas de flujo de los métodos simular, calcula_ruta y recalcula_ruta. Para más detalles consulte el código disponible en el anexo**.**

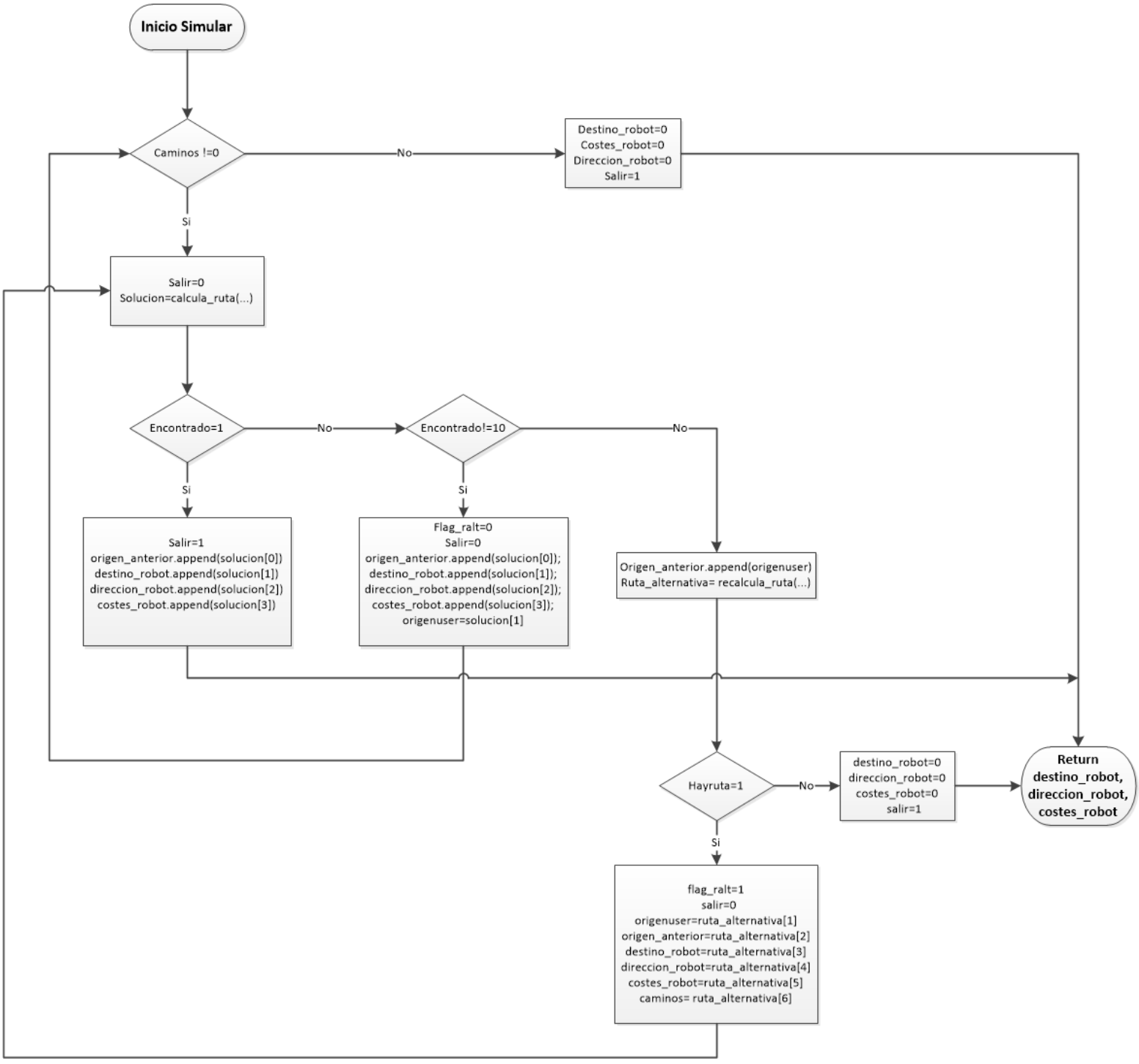

Ilustración 51: Diagrama de Flujo del método simular

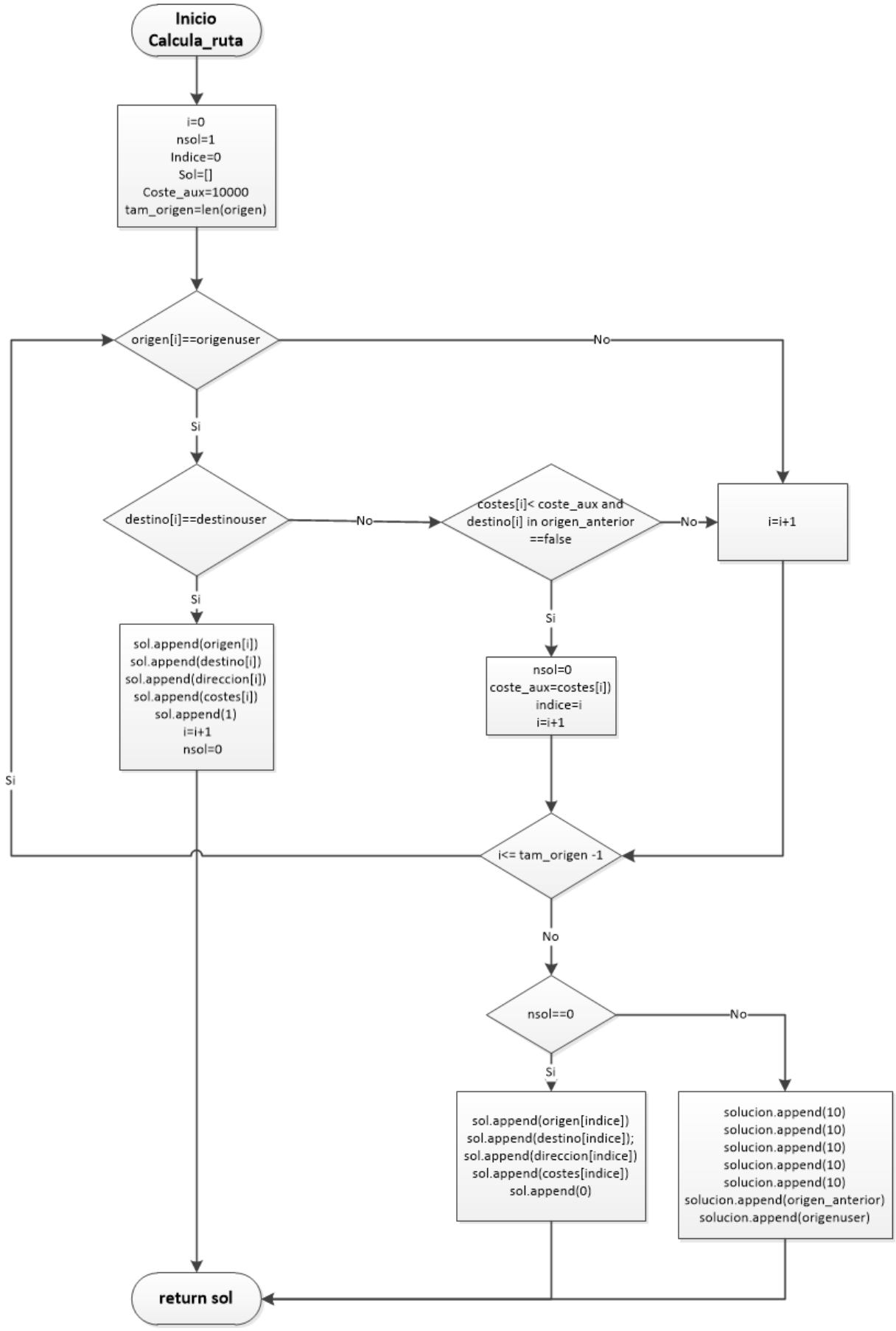

Ilustración 52: Diagrama de flujo del método calcula_ruta

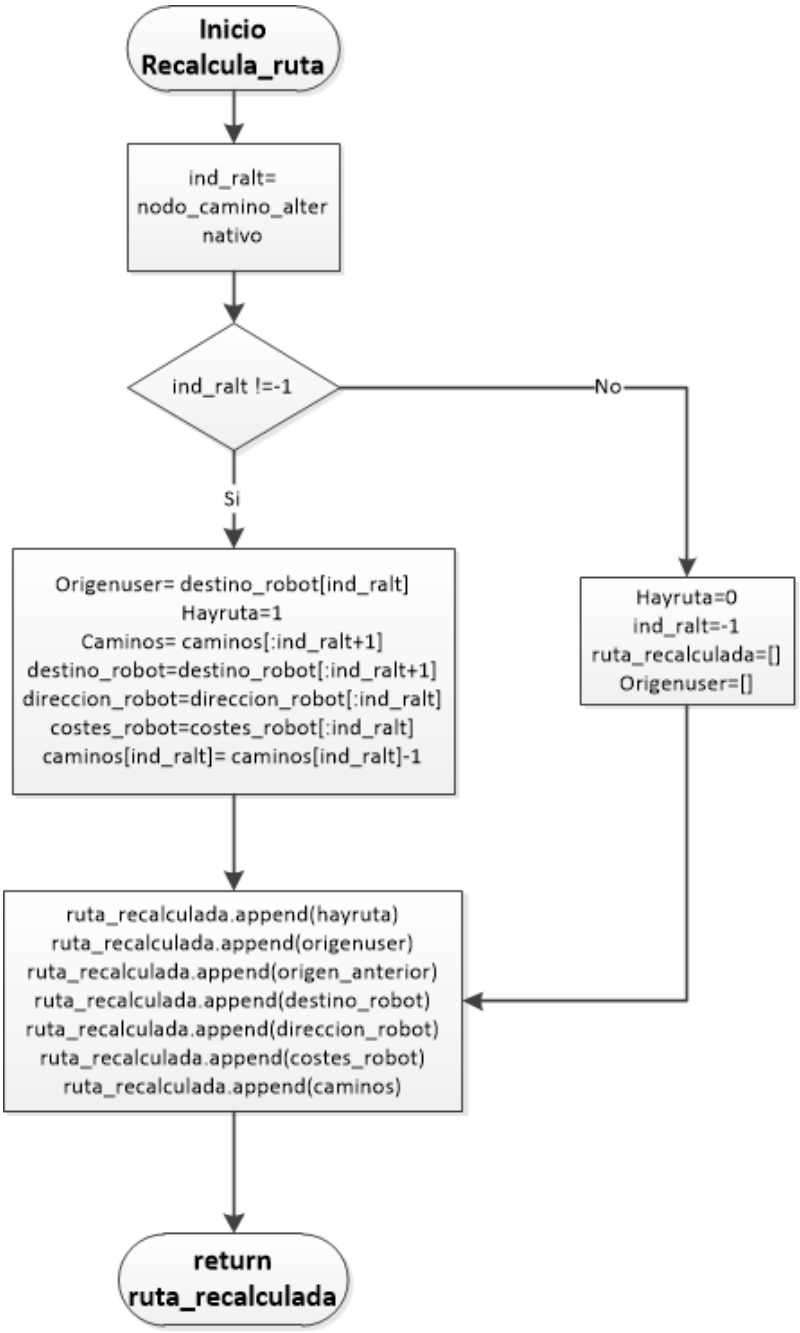

Ilustración 53: Diagrama de flujo del método recalcula_ruta

5.2.6 Método de control remoto del vehículo robot

Este método tiene como objeto dar una alternativa al usuario en caso de no existir un camino entre el origen y el destino. Se sirve de la librería proporcionada por el controlador drv8835, siendo los métodos usados:

- `motors.setSpeeds(m1_speed,m2_speed)` que permite establecer la velocidad y dirección de ambos motores.
- `motors.motors1.setSpeed(speed)` que permite establecer la velocidad y dirección del motor 1.
- `motors.motors2.setSpeed(speed)` que permite establecer la velocidad y dirección del motor 2.
- `time.sleep(tiempo)` que permite mantener una determinada acción durante un tiempo en la forma d0.d1d2d3 segundos.

Cuando se ejecuta esta función se muestra un menú de la siguiente forma:

"Bienvenido al sistema de control remoto del Vehiculo Robot Yowi"
Las opciones disponibles son:
1 para mover el vehiculo hacia delante
2 para realizar un giro hacia la derecha
3 para realizar un giro hacia la izquierda
4 para mover el vehiculo hacia detras
5 para realizar doble giro a la derecha
6 para realizar triple giro a la derecha
7 para realizar doble giro a la izquierda
8 para realizar triple giro a la izquierda
9 para salir del modo control remoto
Seleccione una opcion:

Tras este menú se espera a que el usuario introduzca un valor y presione enter y posteriormente el vehículo realizará el movimiento solicitado (mediante el uso de las funciones anteriores) o saldrá del programa (en caso de pulsar 9). En el anexo puede analizar el comportamiento del programa.

5.2.7 Método de control automático del vehículo robot

Este método actuará siempre que se haya localizado un camino entre el origen y el destino dados por el usuario. La función tiene la siguiente forma:

control_vehiculo(destino_robot,direccion_robot,costes_robot)

recibiendo los destinos que visitará el robot, las actuaciones a tomar y los costes para ir a cada nodo.

En términos generales funciona de forma muy parecida al método de control remoto del vehículo robot con la principal diferencia en que este método ejecuta las acciones del vehículo robot de forma automática. A continuación se presenta el pseudocódigo de este método:

- Mientras que haya direcciones que dar al robot se hace lo siguiente:
    1. Se parte de que el vehículo este parado.
    2. Se calcula el tiempo que el vehículo estará moviéndose.
    3. Se analiza el valor de dirección_robot:
        - Si su valor es 0 se para el vehículo.
        - Si su valor es 1 el vehículo se moverá hacia delante.
        - Si su valor es 2 el vehículo realizará un giro hacia la derecha y continuará hacia delante.
        - Si su valor es 3 el vehículo realizará un giro hacia la izquierda y continuará hacia delante.
        - Si su valor es 4 el vehículo se moverá hacia detrás.
        - Si su valor es 5 el vehículo realizará un doble giro hacia la derecha.
        - Si su valor es 6 el vehículo realizará un triple giro hacia la derecha.
        - Si su valor es 7 el vehículo realizará un doble giro hacia la izquierda.
        - Si su valor es 8 el vehículo realizará un triple giro hacia la izquierda.
        - En caso de tener otro valor se parará el vehículo.
    4. Fin del programa.

Consideraciones importantes:

En el punto dos del pseudocódigo se indica que se calcula el tiempo que el vehículo estará moviéndose. Este tiempo consiste en aplicar la siguiente expresión:

$$t_f = t_{om} * float(costes\_robot[i])$$

donde t_{om} es el tiempo que tarda el vehículo robot en desplazarse un metro (calculado a posteriori) y $costes\_robot[i]$ es la distancia que se tiene que recorrer para llegar al siguiente nodo. El producto de ambos constituye el tiempo que se aplicará sobre los motores.

Cuando se van a efectuar giros no se aplica t_f sobre los motores, en este caso se emplean constantes calculadas: t_d y t_i.

El primero es el tiempo que tarda en efectuar un giro de 90° hacia la derecha y el segundo el propio para la izquierda. Para dobles y triples giros se duplican y triplican respectivamente los valores de dichos tiempos.

En el anexo puede analizar el comportamiento completo del programa.

5.2.8 Programa principal

El programa principal es el encargado de gestionar los datos al comienzo del programa y de realizar las llamadas a los distintos métodos. Cumple con el siguiente pseudocódigo:

1. Preguntar nombre de fichero que contiene la base de datos.
2. Realizar la validación completa de la base de datos (uso de los métodos de validación).
3. Si la base de datos es correcta (sus campos son válidos) se ordena dicha base de datos (uso del método de ordenación).
4. Se recoge del teclado el origen y destino deseados por el usuario.
   - Si el origen es idéntico al destino se indicará que ya se ha llegado al destino y finalizará el programa.
   - Si son diferentes, se pasa la base de datos al programa (uso de los métodos de lectura de la base de datos).
5. Se simula para tratar de localizar un camino entre el origen y el destino solicitados (uso del método de simulación completo).
6. Se revisa el resultado obtenido:
   - Si no existe un camino entre el origen y el destino se informa al usuario y se le da la posibilidad de control remoto del vehículo (uso del método de control remoto).
   - Si existe un camino entre el origen y el destino se convierte a metros los costes (recuerde que la base de datos lo recoge en centímetros) y se ejecuta el método de control automático.

Nota importante: Los datos se proporcionan en la base de datos en centímetros para evitar complicaciones en su gestión.

5.3. Diferencia entre algoritmo Yowi y Dijkstra

La diferencia fundamental entre ambos es que mientras que el algoritmo de Dijkstra necesita un conocimiento global del grafo así como que sea cerrado, el algoritmo Yowi solo necesita tener un conocimiento local del nodo en el que se encuentra para actuar. Otro aspecto a tener en cuenta es que el algoritmo Yowi no garantiza ruta mínima, aspecto que se consigue siempre mediante el algoritmo de Dijkstra.

6 RESULTADOS EXPERIMENTALES

En este apartado se van a detallar los resultados obtenidos desde el punto de vista software y hardware de un problema concreto.

6.1. Planteamiento del problema

Se presenta el siguiente grafo (puede elegirse cualquiera) y se desea demostrar que el algoritmo Yowi es efectivo, es decir, dado un origen y un destino solicitados por el usuario mostrar un camino que permita su union (en caso de existir).

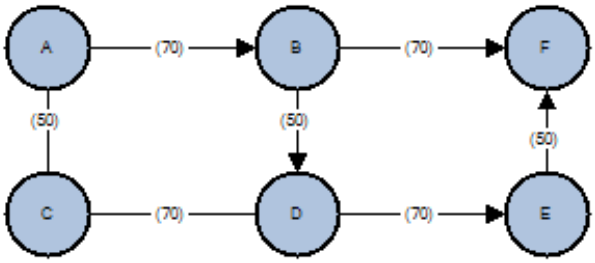

Ilustración 54: Grafo del problema real

Nota importante: Los caminos finalizados por flechas indican la única dirección que se puede tomar mientras que aquellos que no contengan flechas deben ser entendidos como bidireccionales.

6.2. Resultados obtenidos mediante el software

Para obtener el resultado software se debe realizar en primer lugar la implementación de la base de datos en un fichero de texto plano.Se obtiene el siguiente resultado:

Origen	Destino	C0	C1	C2	Dir A	Dir B	Dir C	Dir D	Dir E	Dir F
A	B	0	7	0	1	1	2	1	1	1
A	C	0	5	0	2	2	5	2	2	3
B	F	0	7	0	1	1	1	1	5	5
B	D	0	5	0	2	2	2	2	3	2
C	A	0	5	0	5	2	3	2	2	3
C	D	0	7	0	3	5	1	5	5	5
D	C	0	7	0	5	2	5	5	2	3
D	E	0	7	0	1	3	1	1	3	2
E	F	0	5	0	3	3	3	3	3	2
F	F	0	0	0	0	0	0	0	0	0

Tabla 10: Base de datos del grafo real

Para corroborar el correcto planteamiento de la base de datos se debe tener en cuenta lo siguiente:

1. El vehículo siempre iniciará en posición hacia el este del nodo donde se sitúe.
2. La dirección se corresponderá con la expuesta en el método de control automático del vehículo (véase el apartado 5.2.7 Método de control automático del vehículo robot).

La aplicación del algoritmo Yowi conlleva a los siguientes pasos (supongamos que se desea ir desde A hasta E):

1. Se le solicita al usuario el nombre de la base de datos (el nombre del fichero.txt).
2. Tras introducirlo se comprueba si sus campos son válidos.
3. Una vez superado se le solicita el origen (se introduce A) y destino (se introduce E) deseados.
4. Se comienza a ejecutar el algoritmo cuyos resultados son los siguientes:
   - Solucion inicial: Tomar camino desde A hasta C.
   - Ahora el origen es C.
   - Solucion propuesta: Tomar camino desde C hasta E.
   - Ahora el origen es E.
   - Solución propuesta: Tomar camino desde D hasta E.
   - El programa informa que se ha llegado al destino de la siguiente forma:

```
Ha llegado al destino siguiendo la ruta ['A', 'C', 'D', 'E']
Destinos_robot ['A', 'C', 'D', 'E'] Direccion_robot ['2', '3', '1'] Costes_robot [0.5, 0.7, 0.7]
```

A continuación se ejecuta el sistema de control automático mostrándose de la siguiente forma:

```
Bienvenido al sistema de control del Vehiculo Robot Yowi
Su objetivo es ir desde el origen A hacia el destino E
A continuación se irá desde A hacia C
El vehiculo realizará un giro hacia la derecha
Ya se ha efectuado el giro, ahora continuará avanzando
A continuación se irá desde C hacia D
El vehiculo realizará un giro hacia la izquierda
Ya se ha efectuado el giro, ahora continuará avanzando
A continuación se irá desde D hacia E
El vehiculo se moverá hacia delante
Ha llegado a su destino usando la ruta ['A', 'C', 'D', 'E']
```

6.2.1 Caso particular 1: Necesidad de recalcular la ruta

Supongamos que en el grafo presente en la Ilustración 54 el usuario desea dirigir el vehículo robot desde el nodo origen B hasta el nodo destino A. Por las características del grafo no es posible ir directamente desde B hasta A (el camino es unidireccional y niega el paso de B hacia A). La ejecución del algoritmo puede ser la siguiente:

1. Se le solicita al usuario el nombre de la base de datos (el nombre del fichero.txt).
2. Tras introducirlo se comprueba si sus campos son válidos.
3. Una vez superado se le solicita el origen (se introduce B) y destino (se introduce A) deseados.
4. Se comienza a ejecutar el algoritmo cuyos resultados son los siguientes:
   - Solución inicial : Tomar camino desde B hasta D.
   - Ahora el origen es D.
   - Solución propuesta: Tomar camino desde D hasta E (tenía que haber tomado el camino que lleva de D a C).
   - Ahora el origen es E.
   - Solución propuesta: Tomar camino desde E hasta F (nodo terminal).

En principio el algoritmo finalizaría aqui informando que no hay solución a pesar de existir. Es necesario la aplicación de un algoritmo que permita recalcular la ruta con el objetivo de localizar la solución existente. Se continúa aplicando el método **recalcula_ruta** (véase la Ilustración 53).

5. Se informa lo siguiente: "Se ha localizado un camino que provocará un bucle, se buscará una ruta alternativa para llegar al destino". Probaremos si hay o no ruta alternativa en los nodos visitados [B,D,E,F,F].
6. Se aplica el método **recalcula_ruta:**
   - El método informa de los parámetros que intentará reconducir:

     Los parámetros pasados a recalcula_ruta son:

     Lugares visitados [B,D,E,F,F].

     Caminos hasta el momento [2,2,1,1,1].

     Destinos hasta el momento [B,D,E,F,F].

     "Direcciones hasta el momento [2,3,3,0]".

     Costes hasta el momento [050,070,050,000].

   - A continuación se informa del último nodo que presenta camino alternativo, en concreto: "Hemos localizado que el nodo D tiene más de un camino".
   - Se rectifican los valores de todos los parámetros anteriores prescindiendo de todos aquellos tomados después de haber visitado el nodo D, mostrándose el resultado:

     "Se han realizado las correcciones oportunas para reconducir al robot:"

     "Volveremos a intentarlo desde D".

     "Lugares visitados [B,D,E,F,F]", en este caso se mantiene para no volver a visitarlos.

     "Caminos hasta el momento [2,2]".

     "Destinos hasta el momento [B,D]".

     "Direcciones hasta el momento [2]".

     "Costes hasta el momento [050]".

   - Se devuelven todos los parámetros anteriores para volver a tratar de localizar el camino alternativo.
7. Se vuelve a ejecutar el algoritmo con los datos rectificados:
   - Solución propuesta: Tomar camino desde D hasta C(se toma el camino correcto).
   - Ahora C es el origen.
   - Solución propuesta: Tomar camino desde C hasta A. Se ha alcanzado el destino.

Consideraciones importantes: la aparición de esta posibilidad depende de la forma en que cada usuario introduzca la base de datos (y con ello el orden inicial). No obstante, esto no supone ningún problema ya que (como se ha podido apreciar) queda contemplado en su totalidad por parte del algoritmo Yowi.

6.2.2 Caso particular 2: Control remoto del vehículo robot

Es posible que el usuario introduzca un nodo origen y destino donde no exista ningún camino que los interconecte. Para este caso, el algoritmo da la posibilidad al usuario de tomar el control remoto del vehículo y guiarse por las coordenadas GPS proporcionadas por el otro sistema o parar la ejecución del algoritmo. En concreto, supongamos que en el nodo de la Ilustración 54 el usuario decide que el vehículo robot vaya desde el camino F hasta el camino A:

1. Se le solicita al usuario el nombre de la base de datos (el nombre del fichero.txt).
2. Tras introducirlo se comprueba si sus campos son válidos.
3. Una vez superado se le solicita el origen (se introduce F) y destino (se introduce A) deseados.
4. Se comienza a ejecutar el algoritmo cuyos resultados son los siguientes:
   - Solución inicial: Tomar camino desde F hasta F (F es nodo terminal).
   - Ahora el nodo origen es F.
   - Como F es nodo terminal, no se encuentra ninguna conexión con otro nodo, se tratará de buscar una ruta alternativa (mediante el método **recalcula_ruta**).
   - Como F no tiene camino alternativo se informa al usuario de la siguiente forma: "No se encuentra solución, se evita bucle infinito". "No se ha localizado ninguna solución, se entrará en modo de control remoto".
   - Con ello se muestra un menú con el siguiente aspecto:

Bienvenido al sistema de control remoto del Vehiculo Robot Yowi
Las opciones disponibles son:
1 para mover el vehiculo hacia delante
2 para realizar un giro hacia la derecha
3 para realizar un giro hacia la izquierda
4 para mover el vehiculo hacia detras
5 para realizar doble giro a la derecha
6 para realizar triple giro a la derecha
7 para realizar doble giro a la izquierda
8 para realizar triple giro a la izquierda
9 para salir del modo control remoto
Seleccione una opcion:

   - El usuario decide si realizar el control remoto del vehículo o salir del algoritmo.

6.3. Resultados obtenidos en el vehículo robot

En este apartado se detallarán los pasos desarrollados para la simulación en el mundo real de los resultados obtenidos por el software.

6.3.1 Objetivo inicial: Realización del terreno uniforme libre de obstáculos

El primer paso consiste en llevar a la realidad el grafo mostrado en la Ilustración 54. Para ello, se debe tener en cuenta los siguientes aspectos:

- Las medidas de t_{om} , t_d y t_i variarán en función del terreno.
- El terreno puede prestar más facilidades al movimiento o puede dificultarlo (puede ser resbaladizo).

Conclusión: Realizar la simulación real en un terreno del mismo material, que presente las mismas condiciones en todos los nodos del grafo.

En este proyecto se ha realizado el siguiente terreno basado en material compuesto de "Etilvinilacetato" conocido de forma habitual como goma EVA:

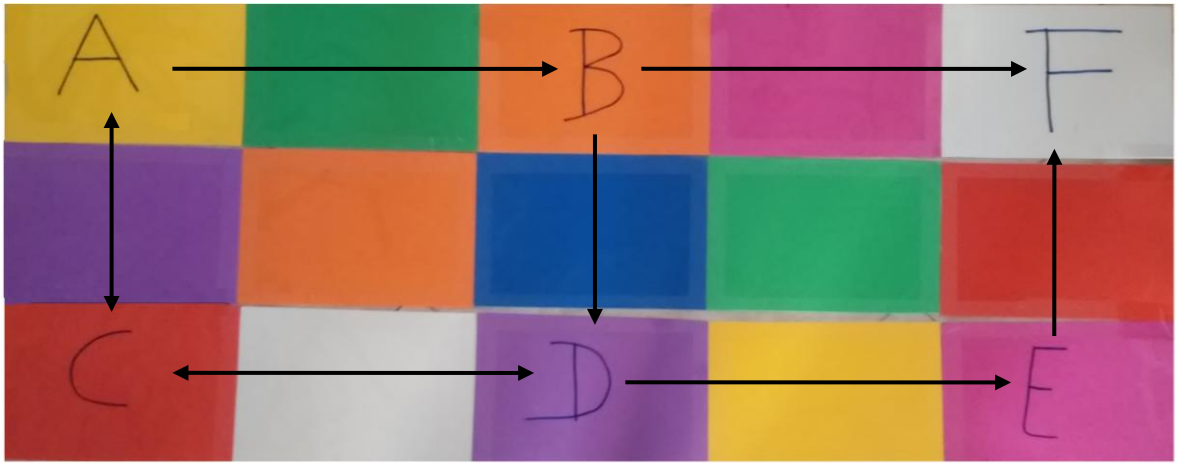

Ilustración 55: Grafo real fabricado con material basado en "Etilvinilacetato"

Cada placa mide 20x30(ambos en cm). La separación entre nodos en horizontal es de 70cm y en vertical de 50cm. El terreno ocupa un total de 60 x 150 (ambos en cm).

6.3.2 Comportamiento del vehículo robot en el terreno

Previo a la aplicación del algoritmo sobre el vehículo robot era necesario obtener los coeficientes t_{om} , t_d y t_i , donde el primero se corresponde con el tiempo que el vehículo tarda en recorrer un metro y los dos últimos se tratan de los tiempos que el vehículo necesita para realizar los giros de 90° hacia la derecha e izquierda respectivamente.

Tras realizar varias pruebas se obtuvieron los siguientes valores:

- $t_{om} = 0.0033$, considerado por la librería como 3.3 segundos.
- $t_i = 0.0037$, considerado por la librería como 3.7 segundos.
- $t_d = 0.0040$, considerado por la librería como 4 segundos.

Es necesario destacar que mientras que para los movimientos hacia delante, hacia atrás y giro hacia la derecha el vehículo se comportaba según lo esperado, no fue así durante las pruebas en el giro hacia la izquierda.

En concreto, el giro hacia la izquierda provocaba que el robot terminara avanzando ligeramente hacia delante durante la finalización de dicho giro, comportamiento que presenta especial interés debido a que se trata de realizar la misma actuación que en el giro hacia la derecha con la diferencia de cambiar el sentido de giro de ambos motores. Se pensó que este suceso era debido a que el PWM de uno de los motores finalizaba instantes después que el del otro motor por lo que se veía afectado en que el vehículo se moviese hacia delante. Sin embargo, se realizaron cambios como disminuir alternativamente (y conjuntamente) la velocidad de los motores pero se comportaba de igual forma.

Por tanto, fue necesario introducir una actuación correctora que se activa solo cuando el vehículo realice el giro hacia la izquierda. Dicha solución consiste en lo siguiente:

1. Realizar giro hacia la izquierda (se produce el defecto de movimiento hacia delante).
2. Se redirige el vehículo hacia atrás.
3. Se compensa el giro.

Para la realización de los pasos 2 y 3 anteriores fue necesario varias medidas de desplazamiento hacia atrás para calcular el tiempo que el vehículo debía moverse, así como compensar su giro. En concreto, estos tiempos son 0.002 y 0.004 respectivamente.

Con el objeto de demostrar que el funcionamiento del algoritmo ha sido un éxito se presenta el siguiente enlace donde se encuentran vídeos de la ejecución completa del resultado del algoritmo en el vehículo robot:

https://youtu.be/ujD6pjmy6f0

7 CONCLUSIONES Y FUTUROS TRABAJOS

En este proyecto se han tenido que resolver dos problemas: uno software y el correspondiente hardware. El primero de ellos, tras su correcta programación y depuración, es compatible con cualquier grafo, encontrando un camino (siempre que exista) entre un nodo origen y destino dados por el usuario. En concreto, se ha probado en distintos grafos: aquellos donde están todos los nodos interconectados (donde siempre hay un camino entre cualquier origen y destino), en aquellos grafos semejantes al resuelto en este proyecto indicando las distintas actuaciones al vehículo robot (si encontró un camino entre origen y destino) o informando si no existe un camino entre el origen y destino.

No obstante, el salto al mundo real implica obtener resultados no esperados (como los problemas en el giro hacia la izquierda del vehículo) y ,por tanto, tener que aplicar las correspondientes correcciones para que se ajusten a la simulación software.

Como futuros trabajos se destaca la implementación del algoritmo de ruta mínima y la comparación con el algoritmo implementado en este trabajo, adecuación de distintos dispositivos de sensorización que permitan obtener las condiciones del terreno como pueden ser la temperatura, humedad, luminosidad, cámara, detección de obstáculos…

Concluimos, por tanto, que la realización de este proyecto ha dado resultados que cumplen con las expectativas, demostrando el potencial del lenguaje de programación python y de sistemas basados en Raspberry Pi.

BIBLIOGRAFÍA Y REFERENCIAS

[1] Subir Kumar Saha. Introducción a la Robótica,Mc Graw Hill.

[2] https://es.wikipedia.org/.

[3] https://commons.wikimedia.org/wiki/Main_Page.

[4] http://www.robotnik.es/.

[5] http://www.irobot.es/.

[6] http://www.rae.es/.

[7] https://www.iso.org/home.html.

[8] Guillermo Heredia Benot. Transparencias Introducción a la Robótica GITT.

[9] Antonio Barrientos, Luis Felipe Peñin, Carlos Balaguer, Rafael Aracil Santoja. Fundamentos de Robótica 2ª edición, Mc Graw Hill.

[10] Aníbal Ollero Baturone. Robótica. Manipuladores y robots móviles, Marcombo.

[11] Maynard Kong. Investigación de operaciones, Fondo Editorial.

[12] Frederick S. Hillier, Gerard J. Lieberman. Introducción a la investigación de operaciones novena edición, Mc Graw Hill.

[13] Cormen, Thomas H, Leiserson, Charles E, Rivest, Ronald L. Introduction to Algorithms Second Edition, McGraw-Hill.

[14] Brassard, Bratley. Fundamentos de Algoritmia 1ª edición, Prentice Hall.

[15] http://tienda.bricogeek.com/robotica/283-chasis-robot-4x4-con-motores.html.

[16] http://tienda.bricogeek.com/expansiones-raspberry-pi/706-controlador-de-motores-drv8835-para-raspberry-pi-b.html.

[17] http://es.farnell.com/microstack/microstack-gps/m-dulo-gps-80-para-raspberry-pi/dp/2434228?ost=2434228.

[18] https://www.element14.com/community/docs/DOC-69361?CMP=KNC-PS-G-EU- GEN-EDC_PRODUCTS.

[19] http://es.farnell.com/microstack/microstack-base-board/placa-expansi-n-base-adapt-raspberry/dp/2434227?MER=sy-me-pd-mi-acce.

[20] http://image.dfrobot.com/image/data/ROB0003/ROB03-Instruction%20MannualV2.0.pdf.

[21] https://www.pololu.com/file/0J570/drv8835.pdf.

[22] https://www.raspberrypi.org/.

[23] https://github.com/pololu/drv8835-motor-driver-rpi.

[24] https://www.element14.com/community/docs/DOC-68898/l/microstack-gps-accessory-module-board.

[25] https://www.continuum.io/what-is-anaconda.

Índice de Ilustraciones

Índice de Tablas

Anexo: Códigos implementados

Programa Principal

```
Abc=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'];


nombre_fichero=raw_input("Escriba el nombre del Fichero de Texto donde se encuentre el Mapa Completo:");
correcto=validar(nombre_fichero);
correctodos=validar_direccion(nombre_fichero)


if correcto==1 and correctodos==1:
print("Se ordenaran los nodos por orden alfabetico")
print
ordenar(nombre_fichero);


print("Escriba el origen desde'A'hasta 'Z':")
origenuser=raw_input();
print("Escriba ahora el destino desde 'A'hasta 'Z'");
destinouser=raw_input();


if(origenuser in Abc) and (destinouser in Abc):


if origenuser==destinouser:
print("Ya se encuentra en el destino");


else:
raw_input("Se ha generado un fichero cuyo nombre es fichero_ordenado.txt, pulse para transferirlo al programa");


(origen,destino,costes)=lectura('fichero_ordenado.txt');
direccion=lectura_direccion('fichero_ordenado.txt',origenuser);
if direccion !=0: #Si el origen se encuentra en la bbdd de direcciones


(destino_robot,direccion_robot,costes_robot)=
simular(origen,destino,costes,direccion,origenuser,destinouser);


if destino_robot==0 or direccion_robot==0 or costes_robot==0:
print
print("No se ha localizado ninguna solucion, se entrara en modo de control remoto") #No se alcanza el destino
print
```

```
control_remoto();
else:
costes_robot=convierte_float(costes_robot)
print("Destinos_robot %s "%destino_robot + "Direccion_robot %s "%direccion_robot + "Costes_robot %s "%costes_robot);
control_vehiculo(destino_robot,direccion_robot,costes_robot);
else:
print "El origen introducido no se encuentra en la topología"
else:
print("No ha introducido un origen o destino valido.El programa finalizara");
else:
print("Revise el nombre o el contenido del fichero");
```

Métodos de validación de la base de datos

```
def validar(fichero):
  Abc=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'];
  Num_costes=['0','1','1.5','2','3','4','5','6','7','8','9'];
  Num_direccion=['0','1','2','3','4','5','6','7','8'];
  correcto=0;

  try:
    infile = open(fichero, 'r');

  except IOError:
    print("El fichero no se encuentra, compruebe su nombre");
  else:

    for line in infile:
      if(line[0] in Abc):
        correcto=1;
      else:
        correcto=0;
        print("Origen No es Correcto");
        break;
      if(line[2] in Abc):
        correcto=1;
      else:
        correcto=0;
        print("Destino No es Correcto");
        break;
      if(line[4] in Num_costes and line[5] in Num_costes and line[6] in Num_costes):
        correcto=1;
      else:
        correcto=0;
        print("Costes No es Correcto");
        break;
      if(line[8] in Num_direccion):
        correcto=1;
      else:
        correcto=0;
        print("Direccion no es correcto");
        break;
```

```
    infile.close();
    if correcto==0:
        print("El fichero no es correcto, reviselo")
    else:
        print("El fichero es correcto")
  return correcto;
```

```
def validar_direccion(fichero):

  Num_direccion=['0','1','2','3','4','5','6','7','8'];
  correcto=0;
  i=8;
  try:
    infile = open(fichero, 'r');
  except IOError:
    print("El fichero no se encuentra, compruebe su nombre");

  else:
    for line in infile:
      a=len(line)-2;
      i=8;
      while i <= a:
        #print i
        if line[i] in Num_direccion:
          correcto=1;
        else:
          correcto=0;
          return correcto;
        i=i+2;
    infile.close();
    if correcto==0:
      print("El fichero no es correcto, reviselo")
    else:
      print("El fichero es correcto")

  return correcto;
```

Métodos de lectura de la base de datos

```
def lectura(fichero):
  origen=[];
  destino=[];
  coste=[];
  try:
    infile = open(fichero, 'r')
  except IOError:
    print("El fichero no existe,compruebe su nombre");
    return origen,destino,coste;
  else:
    print("Se realizara la lectura del Fichero %s "%fichero)
    for line in infile:
      var=line;
      origen.append(var[0]);
      destino.append(var[2]);
      coste.append((var[4]+var[5]+var[6]));
  # Cerramos el fichero.
    infile.close()
  return origen,destino,coste;
```

```
def lectura_direccion(fichero,origenuser):
   direccion=[];
   Abc=['A','0','B','0','C','0','D','0','E','0','F','0','G','0','H','0','I','0','J','0','K','0' + \
       'L','0','M','0','N','0','O','0','P','0','Q','0','R','0','S','0','T','0','U','0','V','0','W','0','X','0','Y','0','Z'];


   indice= Abc.index(origenuser)+8; #Sabemos que la bbdd de direcciones comienza en la direccion 8 y continua en las posiciones pares
                                    #Ademas este indice ya no cambiara ya que leemos siempre la misma posicion pero de cada linea
   try:
      infile = open(fichero, 'r')
   except IOError:
      print("El fichero no existe,compruebe su nombre");
      return direccion;
   else:
      print("Se realizara la lectura del Fichero %s "%fichero)
      for line in infile:
         var=line;
         tam=len(var)-1;
         if indice <= tam:
            direccion.append(var[indice]);
         else:
            direccion=0;
            infile.close()
            return direccion;
   # Cerramos el fichero.
      infile.close()
   return direccion;
```

Método de ordenación

```
def ordenar(fichero):
  tabc=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'];
  i=0;
  j=-1;
  try:
    infile = open(fichero, 'r');
  except IOError:
    print("El fichero no se encuentra, compruebe su nombre");
  else:
    for line in infile:
      i=i+1; #Contamos las lineas
    infile.close();
    while j!=i:
      j=j+1;
      infile = open(fichero, 'r');
      for line in infile:
        if(line[0]==tabc[j]):
          outfile = open('fichero_ordenado.txt', 'a') .
          outfile.write(line) #Se escribe la linea
          outfile.close()
      infile.close();
      if j==len(tabc)-1:   #Se han alcanzado todas las lineas
        j=i;
```

Método de simulación

```
def simular(origen,destino,costes,direccion,origenuser,destinouser):


   caminos=[];
   destino_robot=[origenuser]; #Lista que almacenara los destinos que visitara el robot comenzando por el origen dado por el usuario
   costes_robot=[];
   direccion_robot=[];
   origen_anterior=[];       #Lista que almacenara los lugares visitados y que no se volveran a visitar
   salir=0;
   flag_ralt=0                  #Bandera ruta alternativa, se pondra a uno para evitar que se cuenten los caminos de un nodo del que ya se ha calculado
   ruta_alternativa=[];          #Lista a la que se entregara la solucion proporcionada por el metodo recalcula_ruta
   tamano_origen= len(origen);


   while(salir==0):


      caminos_aux=cuenta_caminos(origen,origenuser,tamano_origen); #Se cuentan los caminos del nodo en el que se esta actualmente
      if flag_ralt==0:


         caminos.append(caminos_aux);     #Solo se almacenara el numero de caminos del nodo si es un nuevo nodo


      if caminos_aux==0: #Si el nodo no existe
         print("No se puede alcanzar la solucion");
         destino_robot=0;
         costes_robot=0;
         direccion_robot=0;
         salir=1;
      else:
         salir=0;
         print("El numero de caminos es %s "%caminos + "Objetivo: Ir desde %s "%origenuser + "hasta %s "%destinouser + " y los lugares visitados son %s "%origen_anterior);
         raw_input("Pulsa una tecla para continuar...");
         #Llamamos a calcula_ruta
         solucion=calcula_ruta(origen,destino,costes,direccion,origenuser,destinouser,origen_anterior)


         #raw_input("Pulsa para continuar");
         if solucion[4]==1:
            salir=1;
            print("Solucion: Tomar camino desde Origen %s "%solucion[0]+ " hasta el destino %s
```

```
"%solucion[1] + "direccion %s "%solucion[2] + "costes es %s "%solucion[3] + "y encontrado es %s "%solucion[4]);
            print
            origen_anterior.append(solucion[0]);
            destino_robot.append(solucion[1]);
            direccion_robot.append(solucion[2]);
            costes_robot.append(solucion[3]);
            print("Ha llegado al destino siguiendo la ruta %s "%destino_robot);
            print("destino_robot %s"%destino_robot + "direccion_robot %s"%direccion_robot + "y salir es %d "%salir + "origen_anterior %s"%origen_anterior +"costes del robot %s"%costes_robot)
        elif solucion[4] !=10:
            salir=0;
            print("Solucion: Tomar camino desde Origen %s "%solucion[0]+ " hasta el destino %s "%solucion[1] + "direccion %s "%solucion[2] + "costes es %s "%solucion[3] + "y encontrado es %s "%solucion[4]);
            raw_input("Pulse una tecla para continuar");
            flag_ralt=0;
            origen_anterior.append(solucion[0]);
            destino_robot.append(solucion[1]);
            direccion_robot.append(solucion[2]);
            costes_robot.append(solucion[3]);
            origenuser=solucion[1];
        else:
            origen_anterior.append(solucion[6]);
            print("Se ha localizado un camino que provocara un bucle, se buscara una ruta alternativa para llegar al destino");
            print("Probaremos si hay o no ruta alternativa en los nodos visitados %s"%solucion[5]);#Solucion[5] es el origen_anterior
            raw_input("Pulsa para continuar");

ruta_alternativa=recalcula_ruta(caminos,origen_anterior,destino_robot,direccion_robot,costes_robot);
            if ruta_alternativa[0]==1:
                flag_ralt=1;
                salir=0;
                origenuser=ruta_alternativa[1];
                origen_anterior=ruta_alternativa[2];
                destino_robot=ruta_alternativa[3];
                direccion_robot=ruta_alternativa[4];
                costes_robot=ruta_alternativa[5];
                caminos= ruta_alternativa[6];



            else:
```

```
            print("No se encuentra solucion, se evita bucle infinito");
            destino_robot=0;
            direccion_robot=0;
            costes_robot=0;
            salir=1;
  return destino_robot,direccion_robot,costes_robot;
```

```
def calcula_ruta(origen,destino,costes,direccion,origenuser,destinouser,origen_anterior):
  i=0;#Indice para recorrer origen
  nsol=1; #Se activara si no encuentra solucion, suponemos que en principio no la hay
  indice=0; #Se almacenara el indice donde se este el coste minimo de los caminos posibles del nodo
  solucion=[];
  coste_aux=10000;
  tam_origen=len(origen);
  while i <=(tam_origen-1):

    if origen[i]==origenuser: # Se ha localizado una linea del nodo donde estamos
        if destino[i]==destinouser: #Si en el siguiente salto podemos alcanzar el destino, lo damos
          solucion.append(origen[i]);
          solucion.append(destino[i]);
          solucion.append(direccion[i]);
          solucion.append(costes[i]);
          solucion.append(1);#encontrado=1
          i=i+1;
          nsol=0;
          return solucion;
        elif ((int(costes[i]) < coste_aux ) and (destino[i] in origen_anterior)==False): #Solo entraremos si el coste es el menor y si aun no se ha visitado el nodo al que vamos a ir
          nsol=0;
          coste_aux=int(costes[i]);#Con esto garantizamos el camino minimo desde el nodo
          indice=i;
          i=i+1;


        else:
          i=i+1; #Si no hemos dado con el destino o destino de coste minimo desde el nodo, buscamos en la siguiente posicion




    else:
      i=i+1; #Si no se ha localizado una linea donde se encuentre el nodo donde estamos buscamos
```

```
en la siguiente posicion


   if nsol==0: #Se ha encontrado un camino hacia otro nodo
      solucion.append(origen[indice]); #Se guarda el origen,destino,direccion y coste y se indica que aun no se ha llegado al destino
      solucion.append(destino[indice]);
      solucion.append(direccion[indice]);
      solucion.append(costes[indice]);
      solucion.append(0); #encontrado=0
   else:      #Se han pasado por todos los caminos de este nodo
      solucion.append(10);#Se ponen origen,destino,direccion,coste y la variable que indica si se ha llegado al destino (encontrado) con valor 10
      solucion.append(10);#Esto permitira avisar de que no se encuentra una solucion y que habra que recalcular ruta
      solucion.append(10);
      solucion.append(10);
      solucion.append(10);
      solucion.append(origen_anterior); #Devolvemos los nodos visitados
      solucion.append(origenuser);  #Devolvemos el ultimo nodo visitado para añadirlo como ultimo
                                    destino visitado
   return solucion;
```

```
def recalcula_ruta(caminos,origen_anterior,destino_robot,direccion_robot,costes_robot):


   origenuser=[];
   i=len(caminos)-1;
   ind_ralt=-1; #Indice ruta alternativa
   hayruta=0; #Indicara si hay o no ruta, en principio no la hay
   ruta_recalculada=[]; #La solucion que proporciona nuestro algoritmo se recogera aqui


   raw_input("Los parametros pasados a recalcula_ruta son:");
   print("Lugares visitados %s "%origen_anterior)
   print("Caminos hasta el momento %s "%caminos)
   print("Destinos hasta el momento %s "%destino_robot)
   print("Direcciones hasta el momento %s "%direccion_robot)
   print("Costes hasta el momento %s "%costes_robot);
   raw_input("Pulse para continuar");
   i=len(caminos)-1;


   while i >=0: #Comprobar(desde el ultimo nodo) si algun nodo visitado tenia ruta alternativa
      if(caminos[i] >1):    #Si algun camino anterior tenia ruta alternativa
```

```
        ind_ralt=i;      #Cual es su indice para localizar ese nodo
        hayruta=1;
        i=-1;
       i=i-1;

   if(ind_ralt !=-1): #Se ha encontrado una ruta alternativa
    origenuser=destino_robot[ind_ralt];  #Volvemos al nodo para tirar por el otro camino
    hayruta=1;
    print("Hemos localizado que el nodo %s tiene mas de un camino "%destino_robot[ind_ralt])
    raw_input("Pulse para continuar")
    caminos=caminos[:ind_ralt+1];
    destino_robot=destino_robot[:ind_ralt+1];
    direccion_robot=direccion_robot[:ind_ralt];
    costes_robot=costes_robot[:ind_ralt];
    raw_input("Se han realizado las correcciones oportunas para reconducir al robot:");
    print("Volveremos a intentarlo desde %s "%origenuser);
    print("Lugares visitados %s "%origen_anterior)
    print("Caminos hasta el momento %s "%caminos)
    print("Destinos hasta el momento %s "%destino_robot)
    print("Direcciones hasta el momento %s "%direccion_robot)
    print("Costes hasta el momento %s "%costes_robot);
    caminos[ind_ralt]= caminos[ind_ralt]-1;
    raw_input("Pulse para continuar");


   ruta_recalculada.append(hayruta);
   ruta_recalculada.append(origenuser); #Contendra el origen desde donde se tratara de buscar una
nueva ruta
   ruta_recalculada.append(origen_anterior); #Aqui se dara a conocer los nodos visitados anteriores
y que no debe volver a visitar
   ruta_recalculada.append(destino_robot); #destino_robot recalculado, se han borrado los destinos
que no han servido
   ruta_recalculada.append(direccion_robot);
   ruta_recalculada.append(costes_robot);
   ruta_recalculada.append(caminos);#Realimentamos el numero de caminos


   print("La ruta recalculada es %s "%ruta_recalculada);
   raw_input("Pulse una tecla para continuar");
   return ruta_recalculada
```

Método de control remoto del vehículo

```
def control_remoto():
  salir=0;
  test_forward_speeds = list(range(0, MAX_SPEED, 1)) + \
    [MAX_SPEED] * 200 + list(range(MAX_SPEED, 0, -1)) + [0]

  test_reverse_speeds = list(range(0, -MAX_SPEED, -1)) + \
    [-MAX_SPEED] * 200 + list(range(-MAX_SPEED, 0, 1)) + [0]
  tam=len(test_forward_speeds)

  while salir==0:

    motors.setSpeeds(0, 0)
    print("Bienvenido al sistema de control remoto del Vehiculo Robot Yowi")
    print("Las opciones disponibles son:")
    print("1 para mover el vehiculo hacia delante");
    print("2 para realizar un giro hacia la derecha");
    print("3 para realizar un giro hacia la izquierda");
    print("4 para mover el vehiculo hacia detras")
    print("5 para realizar doble giro a la derecha");
    print("6 para realizar triple giro a la derecha");
    print("7 para realizar doble giro a la izquierda");
    print("8 para realizar triple giro a la izquierda");
    print("9 para salir del modo control remoto");
    print("Seleccione una opcion:")
    opcion=raw_input();

   if(opcion=="0"):
      print("El vehiculo se parara")
      print
      motors.setSpeeds(0, 0)

    elif(opcion=="1"):
      print("El vehiculo se movera hacia delante")
      print

      for s  in range(0,tam,1):#test_reverse:
        motors.motor1.setSpeed(test_reverse_speeds[s])
        motors.motor2.setSpeed(test_reverse_speeds[s])
        time.sleep(0.0032)
```

```
elif(opcion=="2"):
    print("El vehiculo realizara un giro hacia la derecha");
    print

    for s  in range(0,tam,1):#test_left:
        motors.motor1.setSpeed(test_reverse_speeds[s])
        motors.motor2.setSpeed(test_forward_speeds[s])
        time.sleep(0.0032)

elif(opcion=="3"):
    print("El vehiculo realizara un giro hacia la izquierda");
    print

    for s  in range(0,tam,1):#test_right:
        motors.motor1.setSpeed(test_forward_speeds[s])
        motors.motor2.setSpeed(test_reverse_speeds[s])
        time.sleep(0.0032)

elif(opcion=="4"):
    print("El vehiculo se movera hacia detras");
    print

    for s  in range(0,tam,1):#test_forward_speeds:
        motors.motor1.setSpeed(test_forward_speeds[s])
        motors.motor2.setSpeed(test_forward_speeds[s])
        time.sleep(0.0032)

elif(opcion=="5"):
    print("El vehiculo realizara un doble giro hacia la derecha");
    print

    for s  in range(0,tam,1):#test_left:
        motors.motor1.setSpeed(test_reverse_speeds[s])
        motors.motor2.setSpeed(test_forward_speeds[s])
        time.sleep(2*0.0032)



elif(opcion=="6"):
    print("El vehiculo realizara un triple giro hacia la derecha");
```

```
        print

        for s  in range(0,tam,1):#test_left:
            motors.motor1.setSpeed(test_reverse_speeds[s])
            motors.motor2.setSpeed(test_forward_speeds[s])
            time.sleep(3*0.0032)

    elif(opcion=="7"):
        print("El vehiculo realizara un doble giro hacia la izquierda");
        print

        for s  in range(0,tam,1):#test_right:
            motors.motor1.setSpeed(test_forward_speeds[s])
            motors.motor2.setSpeed(test_reverse_speeds[s])
            time.sleep(2*0.0032)

    elif(opcion=="8"):
        print("El vehiculo realizara un triple giro hacia la izquierda");
        print

        for s  in range(0,tam,1):#test_right:
            motors.motor1.setSpeed(test_forward_speeds[s])
            motors.motor2.setSpeed(test_reverse_speeds[s])
            time.sleep(3*0.0032)

    elif(opcion=="9"):
        print("Ha seleccionado salir del modo control remoto");
        salir=1;

    else:
        print("Introduzca una opcion valida, el vehiculo se parara")
        print
        motors.setSpeeds(0, 0)

return
```

Método de control automático del vehículo

```
def control_vehiculo(destino_robot,direccion_robot,costes_robot):

  i=0;
  tom=0.0033; #Tiempo que tarda en recorrer un metro
  tf=0;     #Tiempo que el vehiculo estara circulando teniendo en cuenta el coste[i]
  ti=0.0037;  #Tiempo que tarda en girar a la izquierda
  td=0.004;  #Tiempo que tarda en girar a la derecha
  test_forward_speeds = list(range(0, MAX_SPEED, 1)) + \
   [MAX_SPEED] * 200 + list(range(MAX_SPEED, 0, -1)) + [0]

  test_reverse_speeds = list(range(0, -MAX_SPEED, -1)) + \
   [-MAX_SPEED] * 200 + list(range(-MAX_SPEED, 0, 1)) + [0]
  tam=len(test_forward_speeds)

  print("Bienvenido al sistema de control del Vehiculo Robot Yowi")
  print("Su objetivo es ir desde el origen %s "%destino_robot[0]+ "hacia el destino %s "%destino_robot[len(destino_robot)-1]);
  raw_input("Pulse para continuar");

  while i <= len(direccion_robot)-1:

    motors.setSpeeds(0, 0)

    tf=tom*float(costes_robot[i]);
    print("A continuacion se ira desde %s "%destino_robot[i]+ "hacia %s"%destino_robot[i+1])
    raw_input("Pulse para continuar");

    if(direccion_robot[i]=="0"):
      print("El vehiculo se parara")
      print
      motors.setSpeeds(0, 0)

    elif(direccion_robot[i]=="1"):
      print("El vehiculo se movera hacia delante")
      print

      for s  in range(0,tam,1):#test_reverse:
        motors.motor1.setSpeed(test_reverse_speeds[s])
        motors.motor2.setSpeed(test_reverse_speeds[s])
```

```
            time.sleep(tf)

      elif(direccion_robot[i]=="2"):
         print("El vehiculo realizara un giro hacia la derecha");
         print

         for s  in range(0,tam,1):#test_right:
            motors.motor1.setSpeed(test_forward_speeds[s])
            motors.motor2.setSpeed(test_reverse_speeds[s])
            time.sleep(td)
         raw_input("Ya se ha efectuado el giro, ahora continuara avanzando")
         for s  in range(0,tam,1):#test_reverse:
            motors.motor1.setSpeed(test_reverse_speeds[s])
            motors.motor2.setSpeed(test_reverse_speeds[s])
            time.sleep(tf)

      elif(direccion_robot[i]=="3"):
         print("El vehiculo realizara un giro hacia la izquierda");
         print

         for s  in range(0,tam,1):#test_left:
            motors.motor1.setSpeed(test_reverse_speeds[s])
            motors.motor2.setSpeed(test_forward_speeds[s])
            time.sleep(ti)
         motors.setSpeeds(0, 0)
         time.sleep(0.003);

raw_input("Correccion")
 for s  in range(0,tam-600,1):#test_reverse:
            motors.motor1.setSpeed(test_forward_speeds[s])
            motors.motor2.setSpeed(test_forward_speeds[s])
            time.sleep(0.002)

motors.setSpeeds(0, 0)
         time.sleep(0.003);

raw_input("Correccion2")
for s  in range(0,tam-580,1):#test_left:
            motors.motor1.setSpeed(test_reverse_speeds[s])
            motors.motor2.setSpeed(test_forward_speeds[s])
            time.sleep(0.004)
         motors.setSpeeds(0, 0)
```

```
    time.sleep(0.003);
    raw_input("Ya se ha efectuado el giro, ahora continuara avanzando")
    for s  in range(0,tam,1):#test_reverse:
      motors.motor1.setSpeed(test_reverse_speeds[s])
      motors.motor2.setSpeed(test_reverse_speeds[s])
      time.sleep(tf)

  elif(direccion_robot[i]=="4"):
    print("El vehiculo se movera hacia detras");
    print

    for s  in range(0,tam,1):#test_reverse:
      motors.motor1.setSpeed(test_forward_speeds[s])
      motors.motor2.setSpeed(test_forward_speeds[s])
      time.sleep(tf)

  elif(direccion_robot[i]=="5"):
    print("El vehiculo realizara un doble giro hacia la derecha");
    print
    for s  in range(0,tam,1):#test_right:
      motors.motor1.setSpeed(test_forward_speeds[s])
      motors.motor2.setSpeed(test_reverse_speeds[s])
      time.sleep(2*td)
    motors.setSpeeds(0, 0)
    time.sleep(0.003);
    raw_input("Ya se ha efectuado el doble giro, ahora continuara avanzando")
    for s  in range(0,tam,1):#test_reverse:
      motors.motor1.setSpeed(test_reverse_speeds[s])
      motors.motor2.setSpeed(test_reverse_speeds[s])
      time.sleep(tf)

  elif(direccion_robot[i]=="6"):
    print("El vehiculo realizara un triple giro hacia la derecha");
    print
    for s  in range(0,tam,1):#test_right:
      motors.motor1.setSpeed(test_forward_speeds[s])
      motors.motor2.setSpeed(test_reverse_speeds[s])
      time.sleep(3*td)

    motors.setSpeeds(0, 0)
    time.sleep(0.003);
```

```
        raw_input("Ya se ha efectuado el triple giro, ahora continuara avanzando")
        for s  in range(0,tam,1):#test_reverse:
            motors.motor1.setSpeed(test_reverse_speeds[s])
            motors.motor2.setSpeed(test_reverse_speeds[s])
            time.sleep(tf)

    elif(direccion_robot[i]=="7"):
        print("El vehiculo realizara un doble giro hacia la izquierda");
        print

        for s  in range(0,tam,1):#test_left:
            motors.motor1.setSpeed(test_reverse_speeds[s])
            motors.motor2.setSpeed(test_forward_speeds[s])
            time.sleep(2*ti)

        motors.setSpeeds(0, 0)
        time.sleep(0.003);
        raw_input("Ya se ha efectuado el doble giro, ahora continuara avanzando")
        for s  in range(0,tam,1):#test_reverse:
            motors.motor1.setSpeed(test_reverse_speeds[s])
            motors.motor2.setSpeed(test_reverse_speeds[s])
            time.sleep(tf)

    elif(direccion_robot[i]=="8"):
        print("El vehiculo realizara un triple giro hacia la izquierda");
        print
        for s  in range(0,tam,1):#test_left:
            motors.motor1.setSpeed(test_reverse_speeds[s])
            motors.motor2.setSpeed(test_forward_speeds[s])
            time.sleep(3*ti)
        motors.setSpeeds(0, 0)
        time.sleep(0.003);
        raw_input("Ya se ha efectuado el triple giro, ahora continuara avanzando")
        for s  in range(0,tam,1):#test_reverse:
            motors.motor1.setSpeed(test_reverse_speeds[s])
            motors.motor2.setSpeed(test_reverse_speeds[s])
            time.sleep(tf)



else:
     print("Introduzca una opcion valida, el vehiculo se parara y se saldra del control del vehiculo")
```

```
        i=len(direccion_robot)
    motors.setSpeeds(0, 0)
    i=i+1;
print("Ha llegado a su destino usando la ruta %s "%destino_robot)
return
```